Scalability analysis of the phase field fracture finite element program

Internal Report No. DLR-IB-SG-AX-2018-251

DLR Deutsches Zentrum für Luft- und Raumfahrt

Scalability analysis of the phase field fracture finite element program

Virtual Engine & Numerical Methods
Physics Based Lifetime Estimation

Start Date: 16.04.2018
End Date: 18.12.2018

Author:
Jonathan Gerlitz
(Deggendorf Institute of Technology)

Supervised and Approved by:
Dr.-Ing. Arun Raina

December 20, 2018

Institute of Test and Simulation for Gas Turbines
German Aerospace Center (DLR e.V.)
Am Technologiezentrum 5
86159 Augsburg

Skalierbarkeitsstudie des auf dem Phasen-Feld Modell basierenden Bruchmechanik Finite Elemente Programms

Virtuelles Triebwerk und numerische Methoden
Physikbasierte Lebensdauerberechnung

Anfangsdatum: 16.04.2018
Abgabetermin: 18.12.2018

Autor:
Jonathan Gerlitz
(Technische Hochschule Deggendorf)

betreut und genehmigt durch:
Dr.-Ing. Arun Raina

Dezember 20, 2018

Institut für Test und Simulation für Gasturbinen
Deutsches Zentrum für Luft- und Raumfahrt
Am Technologiezentrum 5
86159 Augsburg

MASTERARBEIT

Technische Hochschule für angewandte Wissenschaften Deggendorf

Fakultät Maschinenbau und Mechatronik

Studiengang Maschinenbau

Skalierbarkeitsstudie des auf dem Phasen-Feld Modell basierenden Bruchmechanik Finite Elemente Programms

scalability analysis of the phase field fracture finite element program

Masterarbeit zur Erlangung des akademischen Grades:

Master of Engineering (M.Eng.)

vorgelegt von:
B. Eng. Jonathan Gerlitz

Prüfer:
Prof. Dr. Giuseppe Bonfigli

Deggendorf, den 30.11.2018

Erklärung

Name der/des Studierenden: B. Eng. Jonathan Gerlitz

Name des/der Betreuenden: Prof. Dr. Giuseppe Bonfigli

Falls extern: Name des/der Betreuenden: Dr.-Ing. Arun Raina

Thema der Masterarbeit:

Skalierbarkeitsstudie des auf dem Phasen-Feld Modell basierenden Bruchmechanik Finite Elemente Programms

1. Ich erkläre hiermit, dass ich die Masterarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Deggendorf, 30.11.2018 (Datum) Jonathan Gerlitz (Unterschrift der/des Studierenden)

2. Ich bin damit einverstanden, dass die von mir angefertigte Masterarbeit über die Bibliothek der Hochschule einer breiteren Öffentlichkeit zugänglich gemacht wird.

☒ Ja
☐ Nein

Falls Ja:

Ich erkläre und stehe dafür ein, dass ich alleiniger Inhaber aller Rechte an der Masterarbeit, einschließlich des Verfügungsrechts über Vorlagen an beigefügten Abbildungen, Plänen o.ä., bin und durch deren öffentliche Zugänglichmachung weder Rechte und Ansprüche Dritter noch gesetzliche Bestimmungen verletzt werden.

Deggendorf, 30.11.2018 (Datum) Jonathan Gerlitz (Unterschrift der/des Studierenden)

Bei Einverständnis des Verfassers mit einer Zugänglichmachung der Masterarbeit vom Betreuer auszufüllen:

Eine Aufnahme eines Exemplars der Masterarbeit in den Bestand der Bibliothek und die Ausleihe des Exemplars wird

☐ befürwortet
☐ nicht befürwortet.

Deggendorf, _____ (Datum) _____ (Unterschrift des/der Prüfers/in)

Contents

List of Figures

List of Tables

Abstract

In this work, the parallel scalability of the finite element program for the simulation of fracture in elastic-plastic solids is analyzed. The in house finite element program, from here on referred to as PLEANv1.0, is written as a user element and a user material model for the research code Finite Element Analysis Program (FEAP) developed by Prof. R. L. Taylor from the University of California Berkeley. After introducing the fundamentals of the background theory of the finite element method, the phase field method of fracture and the underlying constitutive relations of elastic-plastic solids, the scalability analysis is performed by solving a mechanical problem with 3D unit cube geometry and uniaxial tensile load. The scalability and performance of the PLEANv1.0 is obtained for computations performed over a high performance computing (HPC) cluster using 576 CPUs. A comparison of the scalability and performance of PLEANv1.0 with the research code FEAP for different solution options is also discussed. The promising scalability of PLEANv1.0 allows solution of very large problems in a computationally efficient manner by using the HPC cluster.

1 Introduction

Physics based simulations are becoming more popular by providing higher accuracy and predictive capabilities when compared to the traditional laboratory testing with trial and error method. However, they are becoming more and more demanding in terms of computational memory and time requirements for solving a practical problem. As the design and development of engineering products used in extreme environments such as a jet engine demand really low tolerances, a physics based simulation of the full component on a high performance computing (HPC) cluster becomes a necessity.

Since this requires highly scalable simulation program, commercial as well as open source software developers have changed their programs to a parallelized design. The common simulation platform ANSYS has published the scalability of its applications with respect to multiple problems, different solvers and different clusters. When standard benchmark analyses are performed with ANSYS Mechanical on 8 nodes with 16 processors per node it reaches an efficiency of approximately 40 % [1]. ABAQUS specifies its best performance when standard nonlinear problems are solved. A problem size of $2 \cdot 10^6$ degrees of freedom will yield significant scalability up to 16 processors. Practical large-scale thermo-mechanical simulation with degrees of freedom of the order 10^7 for the thermal and 10^9 for the mechanical problem were computed in [24] where the open source computing platform FEniCS is utilized. Strong scalability was achieved up to 3072 processors.

This work is based on an extended version the research code Finite Element Analysis Program (FEAP) [28] which utilizes PETSc [4] as parallel solver. PETSc's extremely high scalability is represented by the *Gordon Bell Prize* with which one of its applications, a 10^6-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics, was awarded in 2016[1]. The open-source project message-passing interface Open MPI [5] is implemented in FEAP. It enables the program to deploy large-scale problems on large distributed memory systems.

[1] https://www.mcs.anl.gov/petsc/publications/prizes.html

One important research field of physics based simulations is the lifetime estimation of advanced materials and structures which requires accurate prediction of crack initiation and propagation. The phase-field modeling of fracture in elastic-plastic solids as it is described in [10, 16, 17, 18, 19, 20, 21, 22, 23] is provided in form of an in-house program which utilizes libraries of FEAP. It will be referred to as the PLEANv1.0. The high scalability of the PLEANv1.0 program is validated by computing a test problem on a high performance computing (HPC) cluster up to 576 CPUs. A comparison of the scalability and performance of PLEANv1.0 with the research code FEAP for different solution options is also discussed. The promising scalability of PLEANv1.0 allows solving very large problems in a computationally efficient manner by using the HPC cluster.

2 The Continuum Formulation

The fundamentals of the finite element method (FEM) and the underlying continuous formulation will be discussed in this section.

2.1 Strong and weak forms

Consider the body as shown in Fig. 1. It is described by a fixed open domain $\mathcal{B} \subset \mathbb{R}^{n_{\text{dim}}}$ for $n_{\text{dim}} = 1 \dots 3$ dimensions in space. The additional dimension in time is given by $t \in [0, T]$. Each material point of the body is characterized by its position vector $\boldsymbol{x}$ and has a certain density $\rho(\boldsymbol{x}, t)$. Due to the material's density ρ there is a volumetric body force $\mathbf{b} : \mathcal{B} \to \mathbb{R}^{n_{\text{dim}}}$ per unit mass acting on the body $\mathcal{B}$. As shown in Fig. 1, there may be displacements as well as tractions prescribed. Representing the Dirichlet boundary conditions, the displacements $\mathbf{u}_0 : \partial_u\mathcal{B} \to \mathbb{R}^{n_{\text{dim}}}$ are applied to the displacement boundary $\partial_u\mathcal{B} \subset \partial\mathcal{B}$ of the body and there are tractions $\mathbf{t}_0 : \partial_t\mathcal{B} \to \mathbb{R}^{n_{\text{dim}}}$ imposed on the traction boundary $\partial_t\mathcal{B} \subset \partial\mathcal{B}$ which represent the Neumann boundary conditions. The boundary $\partial\mathcal{B}$ of the body is split into displacement boundary $\partial_u\mathcal{B}$ and traction boundary $\partial_t\mathcal{B}$ in a way which follows $\overline{\partial_u\mathcal{B} \cup \partial_t\mathcal{B}} = \partial\mathcal{B}$ and $\partial_u\mathcal{B} \cap \partial_t\mathcal{B} = \emptyset$. We will deal with quasi-static problems where inertial effects are negligible throughout this thesis. Hence we take neither the velocity nor the acceleration field into account. The displacement field is written as $\mathbf{u} : \mathcal{B} \times [0, T] \to \mathbb{R}^{n_{\text{dim}}}$ considering an infinitesimal deformation setting.

Applying the standard gradient operator ∇ in $\boldsymbol{x}$ to the displacement field yields the infinitesimal symmetric strain tensor $\boldsymbol{\varepsilon} : \mathcal{B} \times [0, T] \to \mathbb{R}^{n_{\text{dim}} \times n_{\text{dim}}} = \nabla^s\mathbf{u}$. The symmetric stress tensor $\boldsymbol{\sigma} : \mathcal{B} \times [0, T] \to \mathbb{R}^{n_{\text{dim}} \times n_{\text{dim}}}$ is related to $\boldsymbol{\varepsilon}$ by a constitutive relation characterizing the material response. Symmetry guaranties the satisfaction of the balance of angular momentum.

The strain $\boldsymbol{\varepsilon}$ is additively decomposed into an elastic and a plastic part

$$\boldsymbol{\varepsilon}^e := \boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^p \quad \text{with} \quad \boldsymbol{\varepsilon} = \nabla^s\mathbf{u} \tag{1}$$

where $\nabla^s(\cdot) = \frac{1}{2}\left((\cdot) + (\cdot)^T\right)$. An equivalent plastic strain α is introduced whose time derivative is obtained from the rate of evolution of plastic strain tensor using the $L2$-Norm

$$\dot{\alpha} = \sqrt{\frac{2}{3}}\|\dot{\boldsymbol{\varepsilon}}^p\| \quad \text{with} \quad \dot{\alpha} \geq 0 \tag{2}$$

with the initial condition as

$$\alpha(\boldsymbol{x}, t_0) = 0 \tag{3}$$

With these definitions at hand we are now able to write the strong form of the problem as

$$\left.\begin{aligned} \operatorname{div}\boldsymbol{\sigma} + \rho\mathbf{b} &= \boldsymbol{0} \quad \text{in} \quad \mathcal{B} \\ \boldsymbol{\sigma} &= \mathbb{C}\boldsymbol{\varepsilon} \quad \text{in} \quad \mathcal{B} \\ \mathbf{u} &= \mathbf{u}_0 \quad \text{on} \quad \partial_u\mathcal{B} \\ \boldsymbol{\sigma}\mathbf{n} &= \mathbf{t}_0 \quad \text{on} \quad \partial_t\mathcal{B} \end{aligned}\right\} \tag{4}$$

which represents the following set of equations: balance of linear momentum, constitutive equation and the boundary conditions.

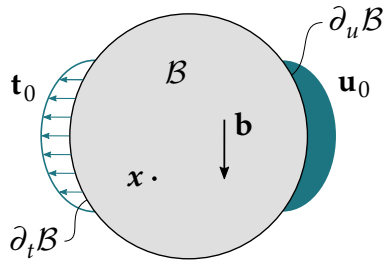


Figure 1: Schematic of a representative body $\mathcal{B}$ with traction $\mathbf{t}_0$ applied to the boundary $\partial_t\mathcal{B}$ and displacement prescribed on the boundary $\partial_u\mathcal{B}$. A body force $\mathbf{b}$ is acting on the body due to gravity.

It is not possible to solve Eq. (4) analytically for all problems with a finite amount of effort. Therefore, we will approximate the solution with the finite element method. We will introduce trial solutions $\mathbf{u}$. It can be shown that the use of polynomials is acceptable in order to describe the displacement field. However these trial solutions are required to satisfy Dirichlet boundary condition per definition in Eq. (4). The trial functions are required to have derivatives that are square-integrable and hence the trial solutions satisfy the definition of H^1 functions

$$\int_{\mathcal{B}} (\nabla \mathbf{u})^2 \, dV < \infty \qquad \text{i.e.} \qquad \mathbf{u} \in H^1 \tag{5}$$

with H^1 being a class of Hilbert space. We will name this first set of function $\mathcal{S}$ and define it as

$$\mathcal{S} = \{\mathbf{u} \mid \mathbf{u} \in H^1,\ \mathbf{u}(\partial_u\mathcal{B}) = \mathbf{u}_0\}. \tag{6}$$

Subsequently we also have to define the variations $\delta\mathbf{u}$ of the trial solutions such that they vanish on the Dirichlet boundary. We introduce them as

$$\mathcal{V} = \{\delta\mathbf{u} \mid \delta\mathbf{u} \in H^1,\ \delta\mathbf{u}(\partial_u\mathcal{B}) = \boldsymbol{0}\}. \tag{7}$$

The set of trial solutions and their variations strongly fulfill the conditions imposed on the Dirichlet boundary.

After defining these two sets of functions we introduce them into the balance of linear momentum Eq. $(4)_1$ and the traction boundary condition. Next we go very briefly through steps of the Galerkin method to reach to the weak form of our problem. Both the balance of linear momentum and the traction boundary condition in Eq. (4) are multiplied with the variations of the trial solutions $\delta\mathbf{u}$ and integrated over their respective domains.

$$\int_{\mathcal{B}} (\operatorname{div}\boldsymbol{\sigma} + \rho\mathbf{b}) \cdot \delta\mathbf{u} \, dV = 0 \quad \text{and} \quad \int_{\partial_t\mathcal{B}} (\boldsymbol{\sigma}\mathbf{n} - \mathbf{t}_0) \cdot \delta\mathbf{u} \, dA = 0 \tag{8}$$

As $\delta\mathbf{u} = 0$ on $\partial_u\mathcal{B}$ we apply the Green's and divergence theorem to (8) and get

$$\left.\begin{aligned} \int_{\mathcal{B}} \operatorname{div}\boldsymbol{\sigma}\cdot\delta\mathbf{u}\,dV + \int_{\mathcal{B}} \rho\mathbf{b}\cdot\delta\mathbf{u}\,dV = 0 \\ \int_{\mathcal{B}} \operatorname{div}(\boldsymbol{\sigma}\,\delta\mathbf{u})\,dV - \int_{\mathcal{B}} \boldsymbol{\sigma} : \nabla(\delta\mathbf{u})\,dV + \int_{\mathcal{B}} \rho\mathbf{b}\cdot\delta\mathbf{u}\,dV = 0 \\ \int_{\partial_t\mathcal{B}} \boldsymbol{\sigma}\mathbf{n}\cdot\delta\mathbf{u}\,dA - \int_{\mathcal{B}} \boldsymbol{\sigma} : \nabla(\delta\mathbf{u})\,dV + \int_{\mathcal{B}} \rho\mathbf{b}\cdot\delta\mathbf{u}\,dV = 0 \end{aligned}\right\} \tag{9}$$

From Eq. (8) we can write

$$\int_{\partial_t\mathcal{B}} \boldsymbol{\sigma}\mathbf{n}\cdot\delta\mathbf{u}\,dA = \int_{\partial_t\mathcal{B}} \mathbf{t}_0\cdot\delta\mathbf{u}\,dA \tag{10}$$

Finally we replace the first term in (9) by the term on the right side in Eq. (10) and rearrange the Eq. (10) to obtain the weak formulation of Eq. (4) in the integral form

$$\int_{\mathcal{B}} \boldsymbol{\sigma} : \nabla(\delta\mathbf{u})\,dV = \int_{\mathcal{B}} \rho\mathbf{b}\cdot\delta\mathbf{u}\,dV + \int_{\partial_t\mathcal{B}} \mathbf{t}_0\cdot\delta\mathbf{u}\,dA \tag{11}$$

Equation (11) is often referred to as the principle of virtual work.

2.2 The phase field modeling of fracture

The deformation of a solid body $\mathcal{B}$ was described in Section 2.1. In order to model fracture in elastic-plastic solids, a phase field based method [12] will be discussed next.

2.2.1 Cracks as diffusive phase field

The crack phase field will be introduced according to [11] considering an infinitely long bar of cross-section Γ with the domain $\mathcal{B} = \Gamma \times L$ where $L = [-\infty, +\infty]$ and the position $x \in L$ of its axis. Assuming a crack at the axial position $x = 0$, the cross-section Γ represents the fully broken crack surface. An auxiliary field variable $d(x) \in [0,1]$ with

$$d(x) := \begin{cases} 1 & \text{for} \quad x = 0 \\ 0 & \text{for} \quad x \neq 0 \end{cases} \tag{12}$$

describes the *sharp crack topology* $d = 0$ represents the unbroken case and $d = 1$ means a fully broken material. This situation is shown in Fig. 2a. The field variable $d(x)$ is introduced as the *crack phase-field*. It is related to the continuum theory of damage, where the development of micro-cracks and micro-voids is described by the scalar damage field d in a homogenized macroscopic sense. In contrast to the sharp crack above the phase field method crack is smeared out over the axial domain L of the bar as

$$d(x) = e^{-|x|/l}. \tag{13}$$

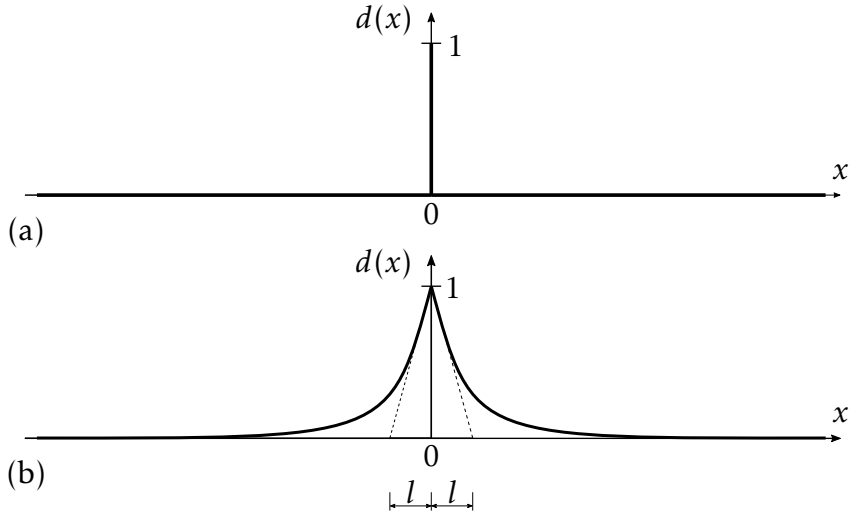


Figure 2: Sharp and regularized crack in the one dimensional case. (a) Sharp crack at location $x = 0$ and (b) regularized crack topology determined by the length scale l.

This *regularized or diffusive crack topology* is determined by the length-scale parameter l. For $l \to 0$ the sharp crack topology of Eq. (12) is obtained. The regularized crack phase field (13) has the characteristics

$$d(0) = 1 \quad \text{and} \quad d(\pm\infty) = 0 \tag{14}$$

Equation (13) is the solution of the homogeneous differential

$$d(x) - l^2 d''(x) = 0 \quad \text{in} \quad \mathcal{B} \tag{15}$$

with Dirichlet boundary conditions as given in Eq. (14). The corresponding variational principle is

$$d = \text{Arg}\left\{ \inf_{d \in W} I(d) \right\} \tag{16}$$

with $W = \{d | d(0) = 1, d(\pm\infty) = 0\}$. The functional $I(d)$ can be obtained by integration of a Galerkin-type weak form of the differential Eq. (15)

$$I(d) = \frac{1}{2} \int_{\mathcal{B}} \{d^2 + l^2 d'^2\} dV \tag{17}$$

Evaluating the functional for the solution (13) yields the identification

$$I(d = e^{-|x|/l}) = l\Gamma_l \tag{18}$$

with $dV = \Gamma_l dx$. By means of this relation of the functional I and the crack surface Γ_l the functional

$$\Gamma_l(d) := \frac{1}{l} I(d) = \frac{1}{2l} \int_{\mathcal{B}} \{d^2 + l^2 d'^2\} dV \tag{19}$$

is introduced. The minimization of this scaled functional yields the diffusive crack topology (13). Important to note is that the length-scale l allows the functional (19) to be considered as the crack surface itself. Therefore the crack surface Γ of the one

Figure 3: Sharp and regularized crack phase field in the multi dimensional case. (a) Sharp crack surface Γ inside the body $\mathcal{B}$. (b) Regularized crack surface $\Gamma_l(d)$.

dimensional bar is achieved when the functional Γ_l is evaluated at the crack location $x = 0$ for a length scale $l \to 0$.

The one dimensional domain of the body can be extended straight forwardly to a n_{dim} dimensional setting. The domain is then given by $\mathcal{B} \subset \mathbb{R}^{n_{dim}}$ with its boundary $\partial\mathcal{B} \subset \mathbb{R}^{n_{dim}-1}$ as shown in Fig. 3. Additionally the dimension in time $T \subset \mathbb{R}$ needs to be considered. The crack phase field may be written as

$$d : \begin{cases} \mathcal{B} \times T \to [0,1] \\ (\boldsymbol{x}, t) \mapsto d(\boldsymbol{x}, t) \end{cases} \tag{20}$$

The multi-dimensional extension of the regularized crack functional (19) reads as

$$\Gamma_l(d) = \int_{\mathcal{B}} \gamma(d, \nabla d) dV \tag{21}$$

with the *crack surface density function* per unit volume

$$\gamma(d, \nabla d) = \frac{1}{2l} d^2 + \frac{l}{2} |\nabla d|^2 \tag{22}$$

as its integrand. Based on a given sharp crack surface topology $\Gamma(t) \subset \mathcal{B}^{n_{dim}-1}$ inside the body $\mathcal{B}$ at time t, as it can be seen in Fig. 3a, the minimization principle

$$d(\boldsymbol{x}, t) = \text{Arg}\left\{ \inf_{d \in W_{\Gamma(t)}} \Gamma_l(d) \right\} \tag{23}$$

yields a regularized crack phase field $d(\boldsymbol{x}, t)$ on $\mathcal{B}$, as shown in Fig. 3b. Here the following Dirichlet boundary condition is applied

$$W_{\Gamma(t)} = \{d \mid d(\boldsymbol{x}, t) = 1 \text{ at } \boldsymbol{x} \in \Gamma(t)\}. \tag{24}$$

The variational principle (23) is equivalent to the Euler equation

$$d - l^2 \nabla d = 0 \quad \text{in} \quad \mathcal{B} \quad \text{and} \quad \nabla d \cdot \boldsymbol{n} = 0 \quad \text{on} \quad \partial\mathcal{B} \tag{25}$$

with Δd being the Laplacian of the phase field and $\boldsymbol{n}$ being the outward normal on $\partial\mathcal{B}$.

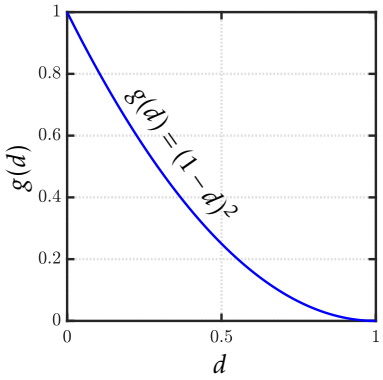


Figure 4: Degradation function $g(d) = (1-d)^2$ as given in [12] to model the transition from the elastic-plastic work density towards the critical value w_c.

2.2.2 Total pseudo work density

We start with an independent observation of plastic and elastic work, followed by the analysis of the total energy as discussed in [12].

With both the elastic and plastic work density as well as the crack surface density function at hand the coupling of plastic deformation to fracture mechanics can be modeled. This is stated in the total work density function W_{tot} which represents the total work needed to deform and crack the solid within the process time.

$$W_{tot} = W_{elas}(\boldsymbol{\varepsilon}^e; d) + W_{plas}(\alpha; d) + W_{frac}(d, \nabla d) \tag{26}$$

$$W_{elas}(\boldsymbol{\varepsilon}^e; d) = g(d)\widetilde{\psi}^e(\boldsymbol{\varepsilon}^e) \quad \text{and} \quad W_{plas}(\alpha; d) = g(d)\widetilde{\psi}^p(\alpha) \tag{27}$$

$$W_{frac}(d, \nabla d) = (1 - g(d))w_c + 2\frac{w_c}{\zeta} l \gamma_l(d, \nabla d) \tag{28}$$

In Eq. (26) it is pointed out that the work being done within a body has three constitutive components: elastic, plastic and fracture. Both the terms for the effective elastic and plastic work density are weighted by the degradation function $g(d)$. It is a monotonically decreasing function which describes the degradation of the elastic-plastic work density due to the evolving fracture. [12] specifies this function as

$$\widehat{g}(d) = (1-d)^2 \tag{29}$$

which is also visualized in Fig. 4. ζ determines the material response after a crack has initiated, i.e. the critical work density w_c - a threshold parameter - has been reached (see Fig. 5).

The effective elastic work density

We first define the effective elastic work density function in terms of the elastic strain measure $\boldsymbol{\varepsilon}^e$ that is computed according to Eq. (1) as

$$\widetilde{\psi}^e(\boldsymbol{\varepsilon}^e) = \frac{\kappa}{2}\mathrm{tr}^2[\boldsymbol{\varepsilon}^e] + \mu \mathrm{tr}\left[\mathrm{dev}(\boldsymbol{\varepsilon}^e)^2\right] \tag{30}$$

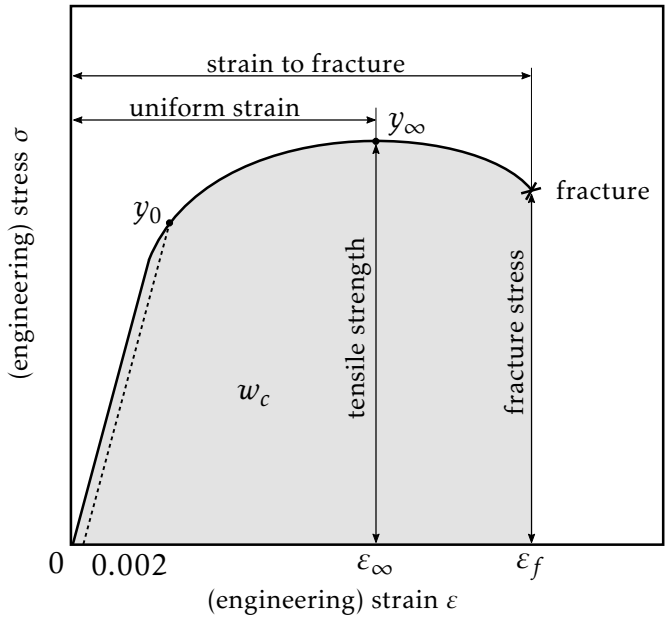


Figure 5: A typical stress-strain curve shows the (offset) yield strength and the saturation yield stress or tensile strength in the context of the tensile test which are relevant for the phase field model [3].

This work density is equivalent to the stored elastic energy of the unbroken material. We note the restrictions that the elastic bulk modulus $\kappa > 0$ and the shear modulus $\mu > 0$.

The effective plastic work density

Moving on from elasticity to plasticity we need to give respect to a number of material parameters that determine the much more complex plastic material response. These parameters can be observed in a standard stress-strain curve for ductile material as it is shown in Fig. 5. The initial (offset) yield stress $y_0 > 0$ prescribes the threshold of the effective elastic response after which plasticity starts. When the load increases further a hardening of the material occurs which causes the stress to continue to increase. This effect is characterized by the isotropic hardening modulus h. After the stress has reached the saturation yield stress y_∞ necking starts. This is followed by a degradation of the material which leads to fracture. At this point the stored energy has reached its critical work density w_c. In addition to these material specific parameters we will introduce the viscosity $\eta > 0$. Thus, we can define an isotropic non-linear hardening function as

$$\widehat{y}(\alpha) = y_0 + (y_\infty - y_0)\big(1 - \exp[-\eta\alpha]\big) + h\alpha \tag{31}$$

which can be found through homogeneous experiments. We can then evaluate the dissipated plastic work per unit volume $\widetilde{\psi}^p$ of the undamaged material by integrating it over the equivalent plastic strain. That means that the $\widetilde{\psi}^p$ can be interpreted as the area under the graph of the function $\widehat{y}(\alpha)$. Therefore, we write the effective plastic

work density of the undamaged material as

$$\widetilde{\psi}^p(\alpha) = \int_0^\alpha \widehat{y}(\tilde{\alpha}) d\tilde{\alpha}. \tag{32}$$

2.2.3 Coupling plasticity to fracture

2.2.4 The fracture evolution

Recapturing that $\Gamma_l(d)$ is the regularized crack surface the first time derivative may be computed in order to obtain an integral for the evolution of cracks. It is also defined as the constitutive crack driving power which is restricted to be non-negative by thermodynamical considerations.

$$\frac{d}{dt}\Gamma_l(d) = \int_{\mathcal{B}} \delta_d \gamma_l(d, \nabla d)\, \dot{d}\; dV := \frac{1}{l}\int_{\mathcal{B}_0} [\,(1-d)\mathcal{H} - \mathcal{R}\,] \cdot \dot{d}\, dV \tag{33}$$

Here a form is used that builds upon the function $\mathcal{R}(\boldsymbol{x}, t)$, that equals the evolution of the phase field, and the function $\mathcal{H}$ which is defined as the maximum of the dimensionless driving state function $\widetilde{D}$. The first function is given by

$$\mathcal{R}(\boldsymbol{x}, t) = \eta \dot{d} \tag{34}$$

The second function gives the dimensionless *crack driving state function*

$$\widetilde{D}(state) = \zeta \left\langle \frac{\widetilde{\psi}^e}{w_c} + \frac{\widetilde{\psi}^p}{w_c} - 1 \right\rangle \tag{35}$$

That means the proportion of effective elastic-plastic energy exceeding the threshold value for damage is weighted with the parameter ζ that models the post critical material response. To restrict the crack driving state function from reversing the crack development the damage criterion is set inside Macauley brackets. The location $\boldsymbol{x}$ from where the crack develops is detected when the maximum crack driving state $\widetilde{D}$ is found. This value yields what is defined as

$$\mathcal{H} = \max_{s\in[0,t]} \widetilde{D}(state(\boldsymbol{x}, s)) \geq 0 \tag{36}$$

and multiplying it by rest damage potential the driving force is formed. When the driving force overcomes the geometrical resistance of the material structure the phase field fracture evolves. Thus, the governing partial differential of the phase field evolution may be written as

$$\eta \dot{d} = (1-d)\mathcal{H} - [\,d - l^2 \Delta d\,] \quad \text{with} \quad \nabla d \cdot \boldsymbol{n}_0 = 0 \;\text{ on } \partial\mathcal{B} \tag{37}$$

where the term on the left side is the evolution which equals the driving force (first term on the right side) when the geometric resistance is subtracted (second term on the right side).

2.3 The Finite element framework

In this section a transition from the continuum framework to the finite element framework will be achieved. It leads to the numerical solution of coupled problem of the balance of linear momentum (4) and the phase field evolution (37).

We discretize the body $\mathcal{B}$ on which Eq. (10) is defined into n_{el} finite number of domains $\mathcal{B}_e^h$ called as finite elements. We want to point out that the discretization means a numerical approximation, because with a finite number of elements, the boundary $\partial\mathcal{B}$ of the body cannot be replicated identically by the boundary $\partial\mathcal{B}^h$ unless an infinite number of elements are used.

Regarding the discretization we introduce shape functions in terms of the local element coordinate system specifically for each node of a finite element. As an example, shape functions are constructed for an 8-node hexadral element [29]. It is chosen because all finite element analysis that are done in this thesis are based on this type of element. The shape functions can generically be written using an index A that may be set to the specific node number

$$\boldsymbol{N}^A = \frac{1}{8}(1+\xi_A\xi)(1+\eta_A\eta)(1+\zeta_A\zeta) \quad \text{with} \quad A = 1\dots 8. \tag{38}$$

By means of the shape functions we can approximate the geometry as

$$\boldsymbol{x} \approx \boldsymbol{x}^h = \sum_{A=1}^{n_{node}} \boldsymbol{N}^A(\boldsymbol{\xi})\boldsymbol{x}_A = \mathbf{N}\boldsymbol{x}. \tag{39}$$

According to the isoparametric concept as it is described in [29] both the geometry and the displacement field can be described using the same shape function. The approximated displacement field becomes

$$\mathbf{u} \approx \mathbf{u}^h(\boldsymbol{x}^h, t) = \sum_{A=1}^{n_{node}} \boldsymbol{N}^A(\boldsymbol{\xi})\boldsymbol{d}_A(t) = \mathbf{N}\mathbf{d} \tag{40}$$

For the numerical solution of the balance of linear momentum (4) it is necessary to define the variation of the displacement field which is obtained by the shape functions

$$\delta\mathbf{u} \approx \mathbf{N}\,\delta\mathbf{d} \tag{41}$$

Since we were able to describe the displacement field based on the nodal displacement vector we can subsequently define the generalized strain field as well by

$$\boldsymbol{\varepsilon} \approx \boldsymbol{\varepsilon}^h(\mathbf{d}) = \sum_{A=1}^{n_{node}} \boldsymbol{B}^A(\boldsymbol{\xi})\boldsymbol{d}_A(t) = \mathbf{B}\mathbf{d} \tag{42}$$

where we introduce the so called strain displacement matrix $\mathbf{B} = \nabla\boldsymbol{x}\boldsymbol{N}$. How this matrix is constructed depends on the type of finite element formulation. Examples are displacement, mixed, assumed and enhanced variational principle.

2.3.1 The discretized form

The weak form of the balance of linear momentum (11) will be discretized. This is followed by a description of its solution procedure. Therefore usage of the strain displacement matrix may be made in order to approximate the gradient of the variation of the displacement field by

$$\nabla(\delta\mathbf{u}) \approx \mathbf{B}\,\delta\mathbf{d} \tag{43}$$

Now $\mathcal{S}$ and $\mathcal{V}$ is discretized, where

$$\mathcal{S}^h \subset \mathcal{S} \qquad \text{(i.e., if } \mathbf{u}^h \in \mathcal{S}^h, \text{ then } \mathbf{u}^h \in \mathcal{S}) \tag{44}$$

$$\mathcal{V}^h \subset \mathcal{V} \qquad \text{(i.e., if } \delta\mathbf{u}^h \in \mathcal{V}^h, \text{ then } \delta\mathbf{u}^h \in \mathcal{V}). \tag{45}$$

By introducing the above described approximations in the weak formulation (11) we obtain

$$\begin{aligned}
\mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\int_{\mathcal{B}_e^h} \boldsymbol{\sigma} : \nabla(\delta\mathbf{u}^h)\, dV \right] &= \mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\int_{\mathcal{B}_e^h} \rho\mathbf{b} \cdot \delta\mathbf{u}^h\, dV + \int_{\partial_t\mathcal{B}_e^h} \mathbf{t}_0 \cdot \delta\mathbf{u}^h\, dA \right] \\
\mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\int_{\mathcal{B}_e^h} (\mathbf{B}\,\delta\mathbf{d})^T : \boldsymbol{\sigma}\, dV \right] &= \mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\int_{\mathcal{B}_e^h} (\mathbf{N}\,\delta\mathbf{d})^T \cdot \rho\mathbf{b}\, dV + \int_{\partial_t\mathcal{B}_e^h} (\mathbf{N}\,\delta\mathbf{d})^T \cdot \mathbf{t}_0\, dA \right] \\
\mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\delta\mathbf{d}^T \cdot \left[\int_{\mathcal{B}_e^h} \mathbf{B}^T\boldsymbol{\sigma}\, dV \right] \right] &= \mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\delta\mathbf{d}^T \cdot \left[\int_{\mathcal{B}_e^h} \mathbf{N}^T\rho\mathbf{b}\, dV + \int_{\partial_t\mathcal{B}_e^h} \mathbf{N}^T\mathbf{t}_0\, dA \right] \right]
\end{aligned} \tag{46}$$

The reader may note that these integrals are defined on the domain of an element and are assembled with an assembly operator. We interpret it as a balance between the sum of the virtual work of internal forces

$$\mathbf{f}_{\text{int}} := \mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\int_{\mathcal{B}_e^h} \mathbf{B}^T\boldsymbol{\sigma}\, dV \right] \tag{47}$$

and the sum of the virtual work of the body forces and surface forces, hence the external forces

$$\mathbf{f}_{\text{ext}} := \mathop{\mathsf{A}}_{e=1}^{n_{el}} \left[\int_{\mathcal{B}_e^h} \mathbf{N}^T\rho\mathbf{b}\, dV + \int_{\partial_t\mathcal{B}_e^h} \mathbf{N}^T\mathbf{t}_0\, dA \right] \tag{48}$$

To define the residual equation we write

$$\mathbf{R}(\mathbf{u}, t) = \mathbf{f}_{\text{ext}}(t) - \mathbf{f}_{\text{int}}(\mathbf{u}, t) = \mathbf{0} \tag{49}$$

where the residuals are given in terms of the nodal displacement vector $\mathbf{u}$.

Equation (49) is nonlinear and is solved with the Newton-Raphson Method. Therefore, we linearize the residuals holding the time constant and choosing the initial dis-

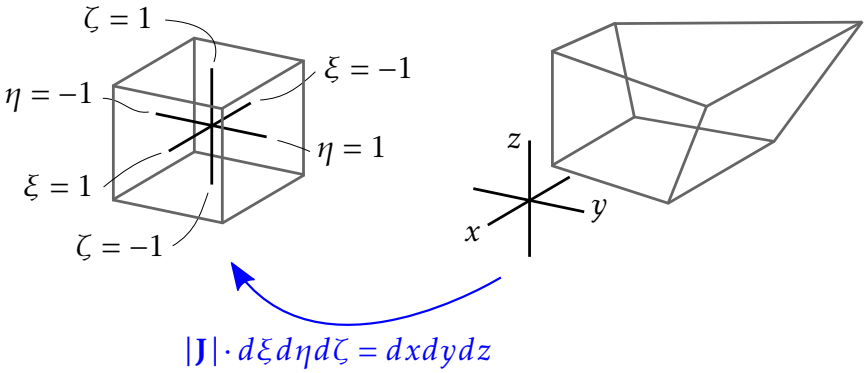


Figure 6: A deformed 8-node brick element in the global coordinate system and undeformed in its parent coordinate system. The integral for the internal forces is solved in the domain as shown on the left side. A coordinate transformation is required.

placements $\mathbf{u}_0$ as a starting point and get the linearized residual as

$$\mathbf{R}(\mathbf{u}) = \mathbf{R}(\mathbf{u}_0) + \frac{\partial \mathbf{R}(\mathbf{u}_0)}{\partial \mathbf{u}} \cdot (\mathbf{u}_0 - \mathbf{u}) \tag{50}$$

Residual vanishes at the state of equilibrium. To find the nodal displacement vector that fulfills this condition, we write Eq. (50) to get

$$\begin{aligned} \mathbf{u} &= \mathbf{u}_0 + \left[\frac{\partial \mathbf{R}(\mathbf{u}_0)}{\partial \mathbf{u}}\right]^{-1} \cdot \mathbf{R}(\mathbf{u}_0), \\ \mathbf{u} &= \mathbf{u}_0 - [\mathbf{K}]^{-1} \cdot \mathbf{R}(\mathbf{u}_0), \end{aligned} \tag{51}$$

where $\mathbf{K}$ is commonly referred to as the stiffness matrix

$$\mathbf{K} := \frac{\partial \mathbf{R}(\mathbf{u}_0)}{\partial \mathbf{u}}. \tag{52}$$

Applying the sequence finding the root of the linearized global problem iteratively we will obtain an acceptable precise solution.

2.4 The internal force array

We have mentioned previously the 8-node hexahedral element is used exclusively to discretize the domain. Therefore we will write all equations in terms of this type of element.

The domain of the body is equivalent with the volume of an element. On the right side of Fig. 6 is a deformed 8-node hexahedron shown in the global Cartesian coordinate system. For various reasons it is convenient to apply a coordinate transformation [29] in order to solve the integral of the internal forces not on the deformed domain but on the undeformed domain in the parent coordinate system instead. The parent coordinate system has the directions ξ, η and ζ all defined on the interval $[-1,1]$. To achieve this change of coordinate system two steps have to be taken. The first one is to simply substitute the infinitesimal volume $dV = dxdydz$ over which the integral is evaluated by the volume of the parent domain which is $dV = d\xi d\eta d\zeta$. Certainly the

volume of the element has changed. Consider this aspect by multiplying the integrand with the Jacobian determinant, defined as

$$|\mathbf{J}| := \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{vmatrix} \tag{53}$$

where we need to calculate a priori the relations between the global coordinate system and the parent coordinate system. We find them as

$$\begin{aligned} \frac{\partial x}{\partial \xi} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \xi} \cdot x_A & \frac{\partial y}{\partial \xi} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \xi} \cdot y_A & \frac{\partial z}{\partial \xi} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \xi} \cdot z_A \\ \frac{\partial x}{\partial \eta} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \eta} \cdot x_A & \frac{\partial y}{\partial \eta} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \eta} \cdot y_A & \frac{\partial z}{\partial \eta} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \eta} \cdot z_A \\ \frac{\partial x}{\partial \zeta} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \zeta} \cdot x_A & \frac{\partial y}{\partial \zeta} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \zeta} \cdot y_A & \frac{\partial z}{\partial \zeta} &= \sum_{A=1}^{8} \frac{\partial \boldsymbol{N}^A}{\partial \zeta} \cdot z_A \end{aligned} \tag{54}$$

These relations contain spatial derivatives of the shape functions. Referring to Eq. (38) where we have defined them for the 8-node hexahedron we may give the partial derivatives as well. They are given by

$$\begin{aligned} \frac{\partial \boldsymbol{N}^A}{\partial \xi} &= \frac{1}{8}\xi_A(1+\eta_A\eta)(1+\zeta_A\zeta) \\ \frac{\partial \boldsymbol{N}^A}{\partial \eta} &= \frac{1}{8}(1+\xi_A\xi)\eta_A(1+\zeta_A\zeta) \\ \frac{\partial \boldsymbol{N}^A}{\partial \zeta} &= \frac{1}{8}(1+\xi_A\xi)(1+\eta_A\eta)\zeta_A \end{aligned} \tag{55}$$

Now we have everything prepared to write the integral of the internal forces in terms of the local element coordinate system by

$$\mathbf{f}_{\text{int}}^{e} = \int_{\zeta=-1}^{1}\int_{\eta=-1}^{1}\int_{\xi=-1}^{1} \left(\mathbf{B}^T \boldsymbol{\sigma}\, |\mathbf{J}|\right) \quad d\xi d\eta d\zeta. \tag{56}$$

This convenient transition to the parent coordinates has yet not made the numerical evaluation of the integral possible. Therefore we apply a numerical integration method which can be implemented in a computer algorithm. According to [9] Gauss quadrature has been established in finite element codes of structural mechanics. This method which is described in [26] evaluates the integrand (56) at n_{GP} points at specific locations $(\xi_i, \eta_i, \zeta_i) \in \mathbb{R}^3$ with $i = 1, ..., n_{GP}$ within the parent domain. The resulting values are multiplied by their respective weights and the summation of these weighted values yields the approximation of the original integral. Hence we compute the inter-

nal forces by

$$\begin{aligned} \mathbf{f}^{e}_{\text{int}} &= \int_{\zeta=-1}^{1} \int_{\eta=-1}^{1} \int_{\xi=-1}^{1} \mathbf{B}^{T} \boldsymbol{\sigma} \, |\mathbf{J}| \quad d\xi \, d\eta \, d\zeta \\ &\approx \sum_{k}^{n^{\zeta}_{GP}} \sum_{j}^{n^{\eta}_{GP}} \sum_{i}^{n^{\xi}_{GP}} \mathbf{B}^{T} \boldsymbol{\sigma} \, |\mathbf{J}| \quad \omega_i \, \omega_j \, \omega_k \end{aligned} \tag{57}$$

3 The Parallel Finite Element Method

As computer models of physical problems become more and more complex also the memory requirements become huge. To obtain results in a reasonable amount of time fast calculations are needed - based on highly scalable software. Therefore, serial computation is not appropriate anymore but high-performance computation which is parallel computation is the answer. In this work parallel computations will be performed on the basis of ParFEAP - the parallel version of FEAP [27]. There are several questions one is faced with when a parallel finite element analysis shall be achieved. The reader is already aware of the fact that parallel computation requires the employment of multiple processors. Here the question arises how the global equation is solved on multiple processors which means that matrix and arrays have to be partitioned. This is followed by the question how each processor knows about what all others are doing in order to find the correct solution for its own partition? We also want to find out how the processors are exchanging information during that process regarding hardware and software requirements. This will give us an answer how the processors are linked together. Another point is how the connection between the processors and the memory is designed. And reaching even deeper into computer architecture we want to understand the basic functionality of a single processor. Regarding performance the very important question is how the performance of a progam is influenced and limited by software and the parallel hardware it is executed on.

3.1 Parallel computer architecture

To start finding answers to the above mentioned questions we first want to discuss the fundamentals of a parallel computer architecture. We start with the structure and functionality of a single processor where we consider the principal structure of a processor in terms of the von Neumann model and then discuss a memory technology that improves the latency of loading data from main memory. From a single processor architecture we then switch to a multi-processor environment. There we discuss following subjects as they are presented in [14]: levels of parallelization, connection between processors as well as between processors and memory and finally cache coherence.

3.1.1 A brief review of the von Neumann model

First of all the principal steps of a processor shall be summarized. This is done on the basis of the *von Neumann model* [13]. When the program is started and therefore loaded into the main memory the control unit goes through four steps. First it needs to fetch an instruction from the memory. Then it decodes it so that as a third step the arithmetic logic unit can execute the instruction. This step includes that data is retrieved from memory and mathematical operations are applied to it. The output is then passed back by the control unit to the memory and is stored there. While instructions and both input and output data are processed they are stored in registers - very low latency memory specifically assigned to the control unit.

Whenever there is an exchange of information between the registers and the memory, the system bus serves for transportation. The bandwidth, i.e. the size of the bus, determines the maximum amount of information that can be passed through within

one clock cycle at which the processor works. The maximum data flow defined by means of bandwidth processor clock cycle is an important measure of performance. Since the development has improved processor performance and decreased the clock cycle time as part of that continuously the required time to load and store data has become more and more a bottle neck.

3.1.2 The implementation of caches to improve loading and storing data

To reduce the access time for data in memory an additional memory has found establishment within processor architecture [8]. Its characteristic property derives from the principle that the bigger the memory the longer it takes to access data. Therefore processor developers have designed a significantly smaller memory, which is called cache, that buffers the retrieved data from the main memory. The buffering yields a second, very meaningful benefit besides the reduced access time due to the small size of the cache. The nature of processing programs is that not all instructions are executed equally often. Therefore those instructions which are frequently or likely to be executed, are primarily stored in the cache. Figure 7 which shows a typical memory hierarchy of a state of the art laptop reveals also that there are multiple cache levels with decreasing size.

The effectiveness of cache management is the miss rate, i.e. how often times the data searched for cannot be found in a cache memory. Two of the possible reasons for a miss are:

- Compulsory - When an instruction or data is retrieved for the first time it cannot be found in cache and gives a miss.
- Capacity - When an instruction or data is retrieved again but is too big to be stored in the cache entirely the rest has to be accessed in the next bigger memory level where it fits. Misses due to an insufficient capacity occur in addition to compulsory misses.

Since the control unit starts always with the smallest cache when it retrieves information (instruction or data) and continues with the next bigger level before it looks in the main memory, a small miss rate is important for the processing performance. Therefore, when there is a set of instruction where most of its parts are reused often times and fit completely into the smallest cache level it will lead to a very low miss rate and yields a high processing performance.

3.1.3 Parallelism within a single processor

Before it is discussed how multiple processors can collaborate to do parallel computation, we first see that even within a single processor work can be done in parallel. When a program is started on a computer the operating system creates a process which executes the program. Nowadays, it can be assumed that all computers are able to do multitasking. This means that a user can run several programs at a time because the processor works for one time slice on one process corresponding to one program and during the next time slice it works on another process. The principle of multitasking can be also applied to a single process. The programmer may form more or less independent parts of a program to *threads*. When one thread is executed and reaches

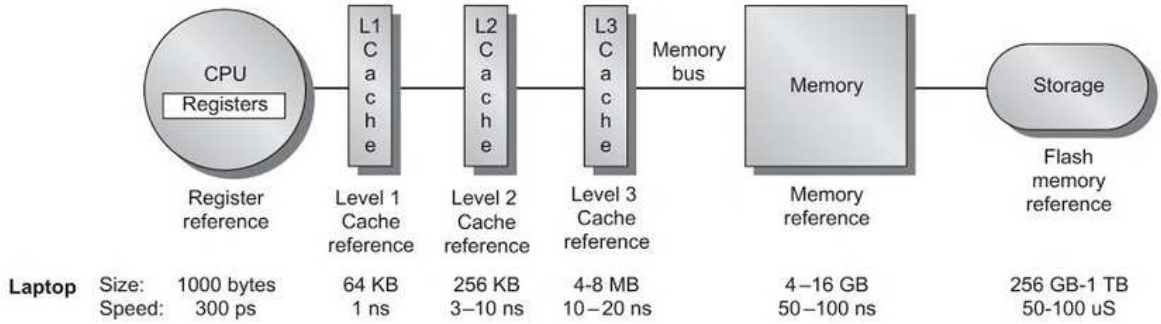


Figure 7: Memory hierarchy and typical storage sizes as it would be found in a laptop. Going down from the disk to the registers memory sizes decrease significantly to enable a fast repeated access of the data [8].

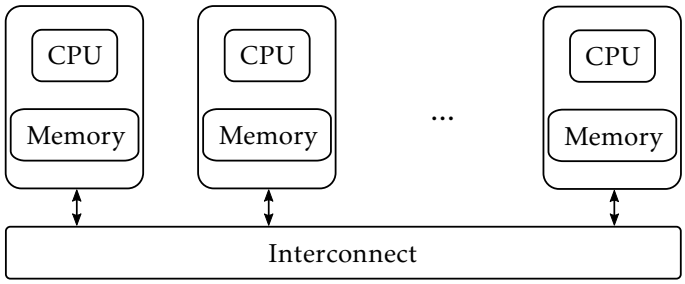


Figure 8: Distributed-memory architecture. Here each processor has its private memory and sends and receives messages from other processor. By communicating with each other the processors exchange their data [14].

a point where it stalls because it has to wait until data is retrieved from memory the process may switch to another thread and progresses on that. This strategy to increase the efficiency of a processor is considered as *hardware multithreading*. When there are within a thread independent instructions they can be distributed to independently working functional units. Due to this the computation time of a single thread can be shortened. This approach is called *instruction-level parallelism*. There are two ways to achieve that. *Pipleline* and *multiple issues*. For more detailed information the reader may refer to [14].

3.1.4 Shared and distributed memory

Although both *instruction-level parallelism* and *thread-level parallelism* is a form of parallel computation it is still based on a single processor. Parallel computing means independently working processors. A parallel architecture distinguishes itself into which extent the memory hierarchy is specifically designated to each processor and what parts may be accessed from all processors. When each of the processors has the full range of memory hierarchy privately available, i.e. it cannot be accessed from another processor, the architecture is called *distributed-memory* systems. This is visualized in Fig. 8. It is shown that the processors need to interconnect in order to exchange information which is stored in the memories that are exclusively designated to the processors. On the other hand systems are called *shared-memory* systems when there is only one main memory implemented that is accessed from all processors. Also

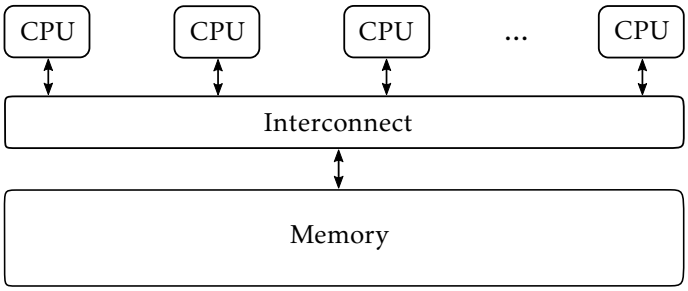


Figure 9: Shared-memory architecture. A parallel system where all processors access the same main memory and interact via sharing its data that is stored in the main memory [14].

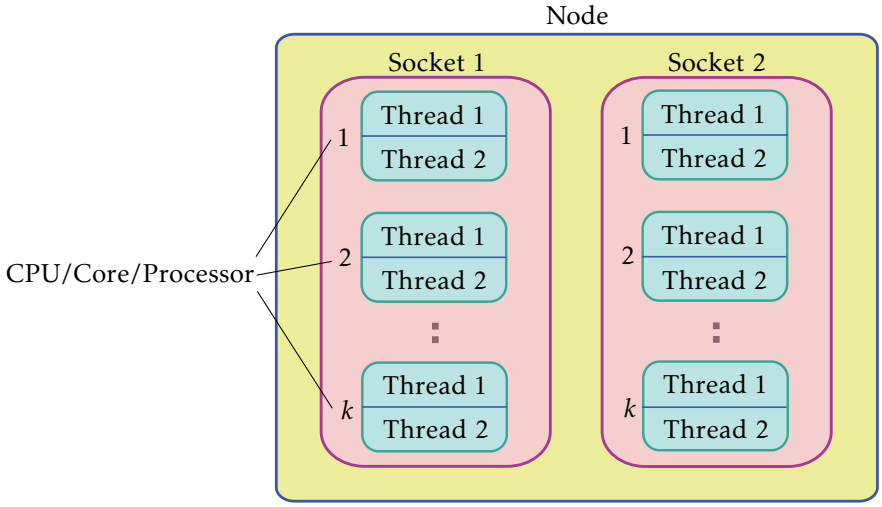


Figure 10: Structure of a compute node.

some cache levels may be shared with at least a number of other processors. Here the processors exchange data implicitly by accessing the same data storage. This model is shown in Fig. 9. A very common design of a *distributed-memory* system is called cluster. Its structure is based on a *distributed-memory* system as shown in Fig. 8 where each of the processor-memory pairs is replaced by a shared memory system - the so called nodes. Without regarding the interconnection or memory organization, Fig. 10 displays the general structure of a compute node with 2 sockets. Each socket contains k processors where each processor is divided into 2 threads. It may be noted that the terms cpu, core and processor are interchangeable.

3.1.5 The interconnection data between processors and memory

There are different designs of how memory is connected with processors. The simplest way of connecting processors with a shared memory is a bus. It can be considered as a pipeline which is accessed by all processors and connected to memory. Buses are a very cost efficient design but bear the disadvantage of becoming a bottleneck due to their finite bandwidth. This is especially the case when the number of processor rises that want to transfer data via the same bus simultaneously.

The crossbar technology is an alternative to a bus system that solves this issue. This technology is visualized in Fig. 11. Multiple processors are connected to indi-

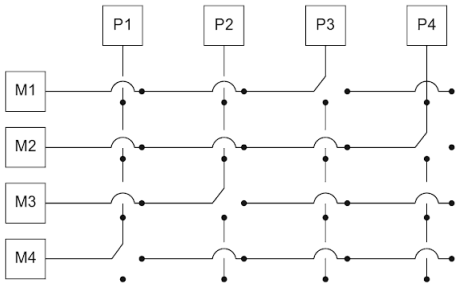


Figure 11: The crossbar interconnection. A grid that exists out of buses and switches interconnects each processor with each module of memory. All processors are most of the times able to load and store data in memory simultaneously [14].

vidual memory modules via a network of buses where each intersection is controlled by a switch. As shown in Fig. 11 it enables each processor to access memory at the same time (with only very few exceptional scenarios). Subsequently the crossbar design yields a much higher performance compared to the bus system especially when the number of processors increases. Both designs are also available to connect the processor-memory pairs of a *distributed-memory* system. The bus system is represented by a ring arrangement. But the same disadvantage that occurs with *shared-memory* systems applies to *distributed-memory* architectures. Therefore, as the number of processors increases it is very convenient to use crossbars instead.

3.2 Performance metrices for parallel systems

Parallel computing is done to achieve an increased performance. Any effort that is taken into that direction has to be measured in order to determine the effort's effectiveness. For that purpose a number of definitions as they are given in [6] shall be discussed.

3.2.1 Run time

The run time of a program that is executed on one or more processors is defined as the range of time starting with the initialization of the program until the (last) processor finishes its work. When we consider a serial system we will refer to the run time by T_s and in terms of a parallel system we will use T_n for it. Run time is representative of "performance" of one algorithm relative to another algorithm.

3.2.2 Speedup

How much faster the parallel run is compared to the serial run is determined by the speedup [6]

$$S = \frac{T_s}{T_n}. \tag{58}$$

For the scalability analysis the run time at the least number of processors can be used as reference instead of the serial run time.

The speedup characteristic shall be explained with the Fig. 12 at hand. First of all three categories can be easily differentiated in Fig. 12a. The blue dashed line repre-

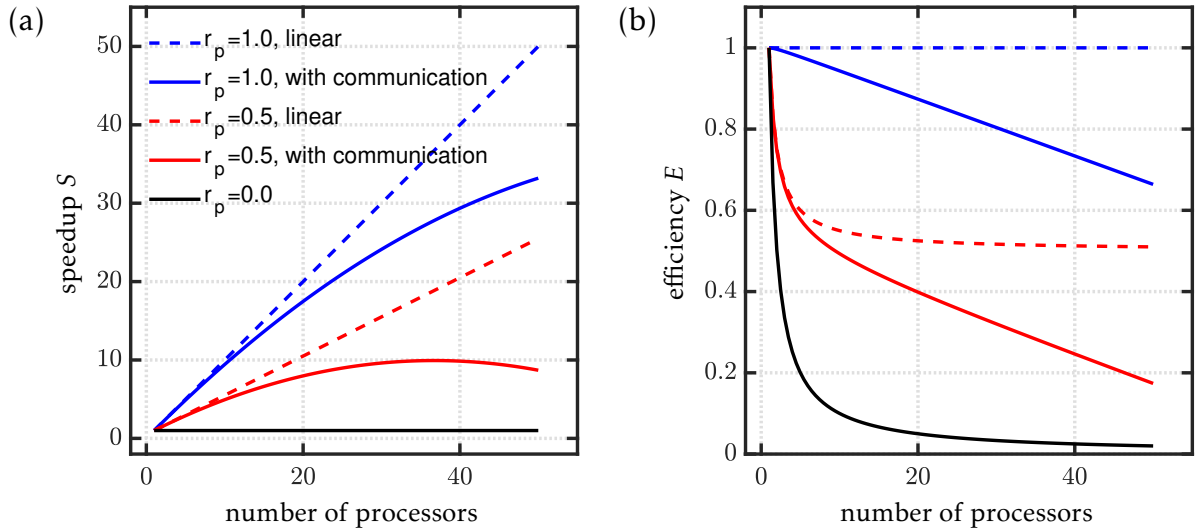
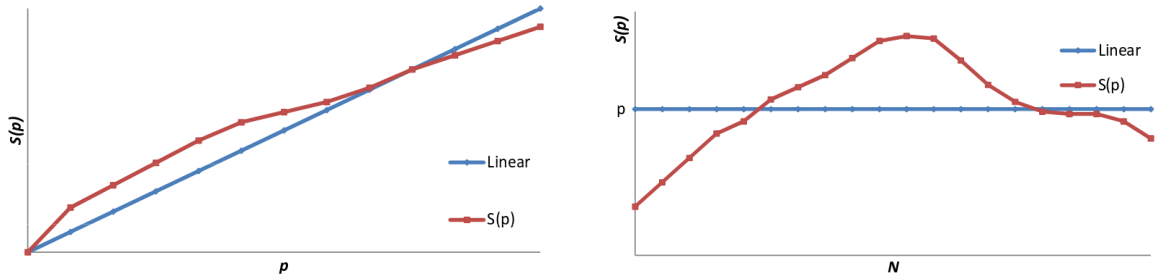


Figure 12: Influence of the software specific parameter r_p and the cost function on performance.

sents the case that when a program is run on n processors instead of the running the best serial algorithm on one processor a speedup of $s = n$ is achieved. This behavior where the speedup curve has a slope of 1 is referred to as *linear* speedup. Does the run time decrease more than the number of processors increases ($s > n$) it is called *superlinear* speedup. In most of the cases there will be a *sublinear* speedup, i.e. $s < n$.

Assuming there is a program given which is entirely parallelized and has a *linear* speedup behavior. The parallel fraction of this code is hence $r_p = 1.0$. Assuming furthermore the total computation time would be doubled by adding serial instructions to the formerly parallel code. The resulting speedup curve $S(r_p = 0.5)$ would be still linear (in mathematical terms) but with only half of the slope.

The linear curves are built on the assumption that there are no additional costs to a parallel solution. Certainly there are additional costs such as the problem partitioning and communication. The partitioning can be assumed to be linear in the number of partitions. The communication cost on the other hand will progressively increase with the number of processors. For both cases, $r_p = 1.0$ and $r_p = 0.5$ respectively was a quadratic communication cost modeled and to the dashed lines applied. It can be observed that at the beginning when the number of processors is still small and therefore the communication cost negligible the resulting solid lines lay on top of the ideal lines. When the number of processors rises the curves flatten out. At some point it may even be the case that the additional communication cost exceeds the amount of time that could have theoretically been saved by an increase in number of processors. Therefore, the slope of the speedup curve becomes negative. The corresponding efficiency curves are shown in Fig. 12b. In [25] two ways are shown how superlinear speedup can be detected. In the first scenario the problem size is kept constant and the number of processors is varied. As shown in Fig. 13a in the region with smaller numbers of processors a superlinear speedup was achieved. This is represented by the red speedup curve laying above the blue linear line. As the number of processors rises the speedup slows down and becomes sublinear. In Fig. 13b is the scenario shown that at a certain number of processors multiple problems with different sizes were run. Here also the



(a) Constant problem size (b) Constant number of processors

Figure 13: Regions of superlinear speedup. Two regions where this effect occurs were detected in [25] (a) if the problem size is constant and the number of processors was varied. (b) Also when the problem size is varied instead and the number of processors remains the same a region for superlinear speedup was detected.

red curve goes clearly beyond the linear curve which means a region of superlinear speedup. [25] gives several explanations for this effect. One important reason relates to the cache organization. Referring to Fig. 13a with smaller numbers of processors the partition sizes are relatively big when the problem size is kept constant. The partition size might require more cache space than available. Therefore when the processor wants to reload the data a cache miss occurs and it looks into higher cache level. If the partition is so big that it does not fit into any cache level the processor finally finds the data it is looking for in the main memory. On the contrary when we increase the number of processors the partition sizes decrease proportionally and at some point will fit entirely into the cache. Subsequently a cache hit occurs which means when the processor wants to reload its partition information it will instantly find it in its private cache memory. This effect saves a significant portion of time that the processor can use to complete its task. Hence it may lead to a *superlinear* speedup.

3.2.3 Efficiency

One disadvantage of measuring the speedup is that one always need to give the number of processes at which it is calculated. Otherwise it cannot be determined whether the given speedup represents a good performance or not. The definition of efficiency solves this issue by simply dividing the speedup S by the number of processes n [6]. Thus we write the formula as

$$E = \frac{S}{n} = \frac{T_s}{n \cdot T_n} \tag{59}$$

Equations (58) and (59) measure the performance *relatively* since they refer to the execution of the same program on a single processor. Thus it is not qualified as an absolute performance measure, i.e. program independent. Therefore for the comparison of two different algorithms the *absolute speedup* and *absolute efficiency* may be defined as

$$S_{abs} = \frac{T_s^*}{T_n} \tag{60}$$

and

$$E_{abs} = \frac{S_{abs}}{n} = \frac{T_s^*}{n \cdot T_n} \tag{61}$$

where the serial run time T_s^* of the best known algorithm becomes the reference [6]. Subsequently the relative speedup and relative efficiency will be identified with the subscript *rel*.

3.2.4 Scalability

Let us suppose we have a program that requires a certain time to complete on a single processor. Furthermore we calculate a certain efficiency. Now we increase the number of processors. If the run time of the program reduces such that the efficiency remains the same we may call the parallel system *scalable*. In other words in a parallel system the run time decreases proportionally to the increase in number of processors. This is represented by linear speedup curve in Fig. 12a.

We implicitly made the assumption that we kept the problem size constant. If that is the case [14] calls this behavior *strongly scalable*. In most of the cases the problem size needs to be increased to keep the efficiency constant. Such systems are called *weakly scalable*.

3.3 Laws of parallelization

On the basis of the definitions we have discussed above we now consider two laws where each gives an understanding for the limits of performance due to parallelization. Each makes different assumptions and consider different parameters. Together these two laws, namely Amdahl's Law and Gustafson's Law, equips us with a broader knowledge of the potential of parallelization.

3.3.1 Amdahl's Law

Amdahl [2] proposed the following formula by which he makes an approach to compute the speedup of a code which has some serial and some parallel regions in it

$$S = \frac{1}{r_s + \frac{r_p}{n}} \tag{62}$$

The portion of the serial part is r_s and r_p stands for the parallel portion. The code may be executed on n processes to increase its performance. When the full code, i.e. $(r_s + r_p)$, is executed on one process let's say it takes one time unit. If we use n processes $1/r_p$ of the total work is split up in n parts. Therefore the parallel solution would require $r_s + r_p/n$ time units. When we simply divide 1 time unit that the serial execution needs by $r_s + r_p/n$ time units according to the definition of speedup in Eq. (58), we get Amdahl's Law as it is formulated in Eq. (62) above.

When we assume that there are no additional costs for the parallel execution, e.g. due to communication, and the entire code could be parallelized the speedup would be proportional to the number of processes. This means that according to Amdahl's Law speedup behavior can be linear at a maximum and never go beyond that to be superlinear. We must conclude that a superlinear speedup is hence only possible due to hardware effects but not due to the software. This correlates to what is published in [25]. This conclusion becomes even more true when we consider a non-zero fraction of code that cannot be parallelized. Increasing the number of processes to infinity we

see that the speedup is limited by the fraction r_s

$$\lim_{n\to\infty} S = \frac{1}{r_s} \tag{63}$$

An important final note on Amdahl's Law is that it is based on the assumption that the total workload that the code handles is constant as the number of processes increases.

3.3.2 Gustafson's Law

On the contrary, John L. Gustafson and E. Barsis in 1988 [7] went into another direction thinking that Amdahl's assumption was not realistic. According to their opinion a user will apply a larger problem when he is given more resources. Therefore they introduced another law, which is referred to as Gustafson's Law, is based on the assumption that the problem size is adjusted to the increased number of processes. And instead of the problem size the run time is kept constant.

They stated that the time required to run the serial fraction of a program does not depend on the problem size as it exists of vector startups, program loading, input and output, etc. Subsequently the parallel run time is proportional to the number of processors or in other words it scales linearly with the number of processors.

Gustafson's Law is like Amdahl's Law a formula to calculate the speedup of a code. But it is based on a different formulation of the parallel run time and therefore leads to a different speedup formulation. The parallel run time T_n is split up in its serial fraction $r_s \cdot T_n$ and its purely parallel fraction $r_p \cdot T_n$. If this code would be executed in a serial mode the serial part of the work would take the same amount of time. The parallel fraction of the work on the other hand would need to be done sequentially instead of at the same time. We therefore have to multiply the parallel fraction by the number of processes and get $r_s \cdot T_n + r_p \cdot T_n \cdot n$ for the serial run time. Implementing these terms in the Eq. (58) for speedup we get Gustafson's Law as

$$S = r_s + r_p \cdot n \tag{64}$$

On the basis of Gustafson's Law and its underlaying assumption that the problem size is adjusted to the increase in number of processes it may be concluded that the speedup is linear in the number of processes.

An increase in the number of processes with a constant problem size means a greater ratio of the serial part of a program in relation to the parallel part one process has to complete. Furthermore the speedup would be sublinear and we would come to the conclusion we have already made regarding Amdahl's Law that the speedup would be limited by the serial fraction r_s. In order to keep that ratio constant which equals to keep the efficiency constant an proportional increase in problem size is required.

3.4 Open MPI: An implementation of the message-passing interface MPI that enables communication between multiple processes

The Message-Passing Interface Forum (MPIF)[2], existing of more than 40 organizations, set a standard for message passing programs in 1994. Their requirements at such

[2]https://www.mpi-forum.org/

programs are defined as to be practical, portable, efficient, and flexible. Since then the standard has seen further development and so already two years later the second version was released. In 2012 then the release of MPI-3 was made [5].

Since MPI is a standard it is not ready to be implemented in a code. To bring the MPI standard up to this level the Argonne National Laboratory developed MPICH[3] which has always been updated to the current MPI version. With MPICH the Argonne National Laboratory follows its goals to make MPI applicable to various computation and communication platforms as well as to provide a framework that can easily be extended. For the latter goal MPICH is freely available. Therefore many implementations of MPI have been developed by different kinds of institutions based on MPICH.

Another open source MPI project is Open MPI. It is a complete implementation of the message-passing interface and provides high performance communication. Open MPI came to existence as three well-known MPI implementations, namely FT-MPI from the University of Tennessee, LA-MPI from Los Alamos National Laboratory and LAM/MPI from Indiana University, were merged together with contributions from the PACX-MPI team at the University of Stuttgart. The purpose of this fusion was to combine their specific excellencies to form one powerful implementation.[4]

As Open MPI has also found implementation in ParFEAP to enable high-performance finite element analysis we want to make some more detailed notes on this library. Open MPI defines functions which can be implemented in a code, e.g. ParFEAP. Therefore any such code must have an include statement at the beginning which links the Open MPI library to it. A description of the syntax and how the functions are to be implemented is found in [15]. Three programming languages are supported in which the basic code may by written in: C, C++, and Fortran. Once the code is fully developed including Open MPI functions it is compiled using a compiler wrapper. Assuming we would use Fortran as programming language the corresponding compiler wrapper is `mpifort` which supports any Fortran version. Then the program may be invoked with the `mpirun` command. Let us imagine there are several processes running at the same time. Each process would be identified via its rank which can be considered as an address. As the processes are interacting with each other we find different categories of communication. In point-to-point communication one processes sends a message to or receives a message from other processes. On the other side it may also communicate with more than one other process which would then be called a collective communication. However in general a sending process may or may not request a confirmation of reception. If a confirmation is required the process which is waiting for the confirmation either is blocked or does some other work in the mean time. Which case applies depends on which functions are implemented in the code.

3.5 PETSc: a library for the parallel solution of partial differential equations

ParFEAP uses PETSc solver libraries for parallel solution. [4] introduces it as a toolkit for the numerical solution of partial differential equations and related problems on high-performance computers. The Portable, Extensible Toolkit for Scientific Computation (PETSc) is designed by the Argonne National Laboratory which is a U.S. Depart-

[3] https://www.mpich.org/
[4] https://www.open-mpi.org/

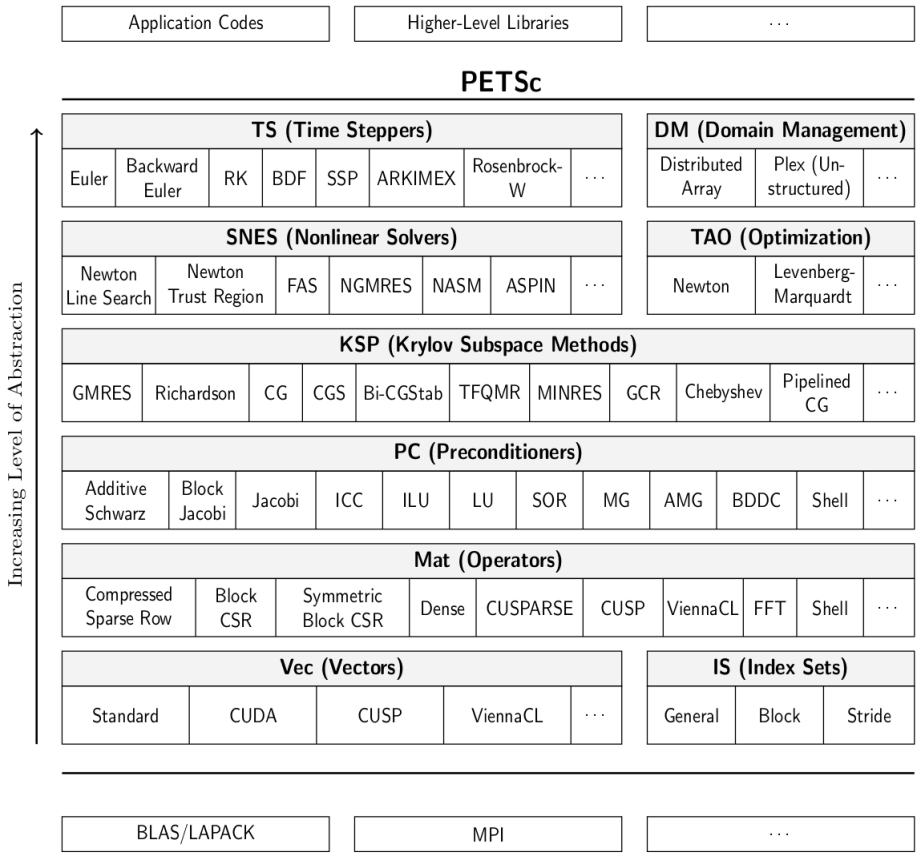


Figure 14: Overview of the PETSc library [4].

ment of Energy. The toolkit includes data structures and routines that interact with a given code in order to solve a scientific problem. The application codes may be written in Fortran, C, C++, or Python (via petsc4py) where especially the object oriented programming style yields an environment for PETSc to show its performance.

Figure 14 gives us an overview of numerical libraries that PETSc provides. Using PETSc one will always start employing objects of the highest level of abstraction. Examples for such objects are Time Steppers and Nonlinear Solvers which are placed at the top of the figure. On the contrary matrices and vectors are considered as elements of the lowest level of abstraction. In Fig. 14 we see above and below the PETSc objects examples given for external software. As already mentioned PETSc routines exchange their information according to the MPI standard. BLAS/LAPACK packages provide subprograms in terms of numerical algebra and high-performance computers. At the very top of the figure it is indicated that other libraries may be used in order to solve a problem with the application code which represents ParFEAP in our case.

3.6 Parallel solution of the FE problem

In this section the parallel solution structure as implemented in ParFEAP [28] to solve the FE problem is discussed.

The goal is that the computation of element arrays, the assembly of global set of equations and the solution of the global problem is done by multiple processors. Therefore by giving the `GRAPH node numd` command in the input file for ParFEAP the

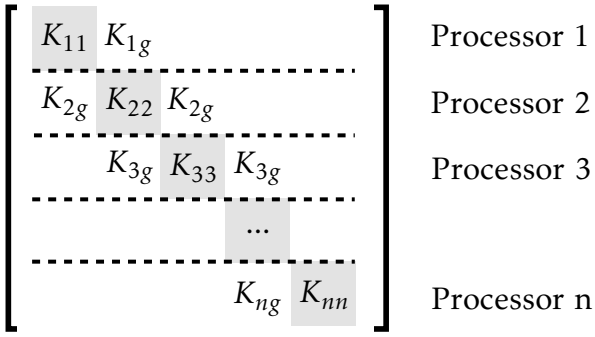


Figure 15: Assembly of the partitioned problem to the global stiffness matrix. [28]

user defines the number of processors. This command will call functions of the library METIS which provides a set of serial algorithms for partitioning the finite element mesh as well as for ordering the matrix in a fill-reducing manner. For very large problems ParMETIS offers these algorithms as a parallel version in combination with other features. Subsequently all nodes that form the mesh are divided into `numd` sections where each partition will be handled by a specific processor. Ideally all partitions contain an equal number of nodes which is calculated as the total number of nodes divided by the number of processors. This will be approximately achieved by METIS as the number of nodes per partition varies slightly. The `GRAPH` command is followed by the `OUTDOMAINS` command which writes a complete description of the mesh, including e.g. the nodal coordinates, into `numd` files. To achieve a good performance during the assembly of the global matrix its required memory space is pre-allocated.

The reason why the number of nodes per partition varies slightly is because of an odd number of processors. But there is also another reason which is related with the solution of the global set of equations. For the sake of communication between nodes of different partitions ghost nodes are generated for each partition. As their name suggests they do not have a geometrical meaning but a purely numerical. Therefore the number of nodes per partition is equal to the sum of nodes and the generated ghost nodes.

The global stiffness matrix existing of all nodes as well as ghost nodes is divided into sections as shown in Fig. 15. All equations in the global problem are numbered from 1 to the total number of equations. then the first set of equations is designated to processor 1, the second set to processor 2, etc. The matrix is arranged such that elements corresponding to nodes are placed in the diagonal block and the elements corresponding to ghost nodes are situated in off-diagonal blocks. According to that arrangement the structure of the displacement vector is defined. During the solution process of the global set of equations PETSc does only save the residuals referring to (real) nodes. The residuals corresponding to ghost nodes are not saved. This strategy requires less communication since only the displacements and not the residuals of the ghost nodes need to be exchanged after each solution step. Having discussed the essential details of the problem description and solution we finally look at how the procedure is initiated. As already mentioned the global problem is represented by an input file. This file is called by ParFEAP in order to partition the mesh and to generate input files for all processors that will be employed. After all input files are created the user issues the `mpirun` command e.g. as

Solver	Preconditioner	Matrix format
CG	Jacobi	AIJ and BAIJ
CG	Hypre with Boomerang	AIJ
CG	ML/Trilinos	AIJ
CG	GAMG	AIJ with BCIN (successor to Prometheus)
MINRES	Jacobi	AIJ and BAIJ
GMRES	Jacobi	AIJ and BAIJ
GMRES	Block Jacobi	Often gives indefinite factor
GMRES	ASM (ILU)	
SuperLU	(direct)	AIJ (has BLAS conflict on Mac OS X $\geq$ 10.7.5)
Spooles	(direct)	AIJ
MUMPS	(direct)	AIJ

Table 1: Combinations of solver, pre-conditioner and matrix format from the PETSc library that are successfully tested with Parallel FEAP. [28]

```
mpirun -np $n $PARFEAP -ksp_type cg -pc_type gamg ⟨other options⟩
```

This command line initiates ParFEAP to compute element arrays, etc., and finally PETSc to solve the global set of equations. On that line the number of processors $np = \$n$ and the directory where the ParFEAP executable `$PARFEAP` is saved is specified. Additionally the user need to choose the solver type, e.g. the conjugated gradient method which belongs to the Krylov Subspace sMethods, and the matrix preconditioner. In the above example the GAMG preconditioner is used. In the manual a list of pre-conditioners, solvers and matrix formats are given that had been tested in combination so far (Tab. 1). Yet there are much more possibilities that PETSc provides. When the PETSc routines are called they will solve the given problem according to the specific solution procedure as it is given by the user in a specific file. There are many ways and options to it, for example for a quasi-static problem the increment in displacement is given there that is applied to the body after each solution step. Also limits for convergence, the maximum number of iterations as well as which values shall be output in the output files are given there. When we reach the point of considering specific problems that has been solved within this thesis we will also discuss their solution procedures in detail.

4 Representative Numerical Simulations

All simulations presented in this section were performed on a HPC cluster with 150 nodes (Tab. 2 and Tab. 4). All nodes exists of 2 sockets each having 12 processors of the type *Intel®Xeon processors E5-2695 v2 @ 2.40GHz* (Tab. 3). The scalability studies presented in the manual of ParFEAP [28] were run on a cluster of *AMD Opteron 250 processors* that are connected together via a Quadrics QsNet II interconnect system.

component	technology	performance/size
150 nodes	2x Intel®Xeon processors E5-2695 v2 @ 2.40GHz	3600 cores peak performance 70.2 TFLOPS
memory	distributed	19.2 TB (aggregate)
shared disk	GPFS, parallel file system	118.7 TB
local disk	SATA (250 GB)	37.5 TB (aggregate)
interconnect	FDR infiniband	56 Gbit/s

Table 2: System configuration and performance details of the cluster used for this work.

component	description
sockets per nodes/core per socket	2/12
CPU	intel®Ivy Bridge E5-2695v2 @ 2.4 GHz
memory per node	64 GB (8x 8 GB) 4 channels DDR3-1866 Mhz
processor interconnect	2x QPI 8.0 GT/s+
PCI Express	40 lanes, Gen 3.0
250 GB disk	1x SATA2 7.2k HDD drive
high-speed network	IB 4x FDR host channel adapter
operating system	SuSE SLES 11 SP3

Table 3: Technical specification of the HPC cluster.

component	description
9x FDR infiniband switchblades	SGIICE X
5x FDR switches 36-port	Mellanox MSX 6025

Table 4: Specification of the network components.

Software packages and libraries that are used to run simulations have to be considered. Their version is noted in Tab. 5.

software package	this work	manual
FEAP	v8.5	v8.5
MPI	v3.1.2	v1.2.4
GCC	v8.2.0	v3.3.4
PETSc	v3.9.2	v2.3.2-p3
AMD ACML (BLAS/LAPACK)	v3.4.2	v3.5.0
Prometheus	n.a.	v1.8.5
ARPACK	2001	2001

Table 5: Comparison of software packages installed on the protec cluster and the cluster on which the analyses of the ParFEAP manual were run.

4.1 Examples of ParFEAP manual

The ParFEAP manual [28] shows different simulations in order to validate scalability. Three examples were chosen and simulated on the available cluster. According to the manual the algebraic multigrid preconditioner Prometheus was utilized for its simulations. Since this preconditioner is not available in PETSc any more its successor GAMG was used. To analyze the scalability behavior the manual refers to the *MFLOPS* values from the KSPSolve object from PETSc. It is defined as the sum of 10^6 floating point operations over all processors divided by the maximum time over all processors

$$MLFOPS := \frac{MFLOP}{time_{nproc,max}} \tag{65}$$

Thus the manual focuses on the scalability of solving the global set of equations. The speedup formula from Eq. (58) has to be rewritten as

$$S = \frac{T_s}{T_n} = \frac{MFLOPS_n}{MFLOPS_s}\,. \tag{66}$$

The efficiency is redefined by

$$E = \frac{S}{n} = \frac{MFLOPS_n}{n \cdot MFLOPS_s} \cdot 100\,\%\,. \tag{67}$$

Since the examples were redone with the identical program the definition of the relative speedup and relative efficiency may used for the validation instead of the absolute metric (see Section 3.2.3).

As shown in Tab. 1 the preconditioner *GAMG* was officially tested only with the Krylov Subspace Method *conjugated gradient* as solver so far. Therefore, this solver was chosen for the reproduction of the examples. For all three examples the load was applied in 100 steps.

4.1.1 Linear elastic block with 8-node brick elements

In the first example a unit block is discretized with 70^3 elements of the type 8-node hexahedron. All bottom nodes are fixed in all coordinate directions and for nodes at the top surface a uniform load in all three directions is applied. The elements of the mesh were associated with the linear elastic isotropic material model from the library

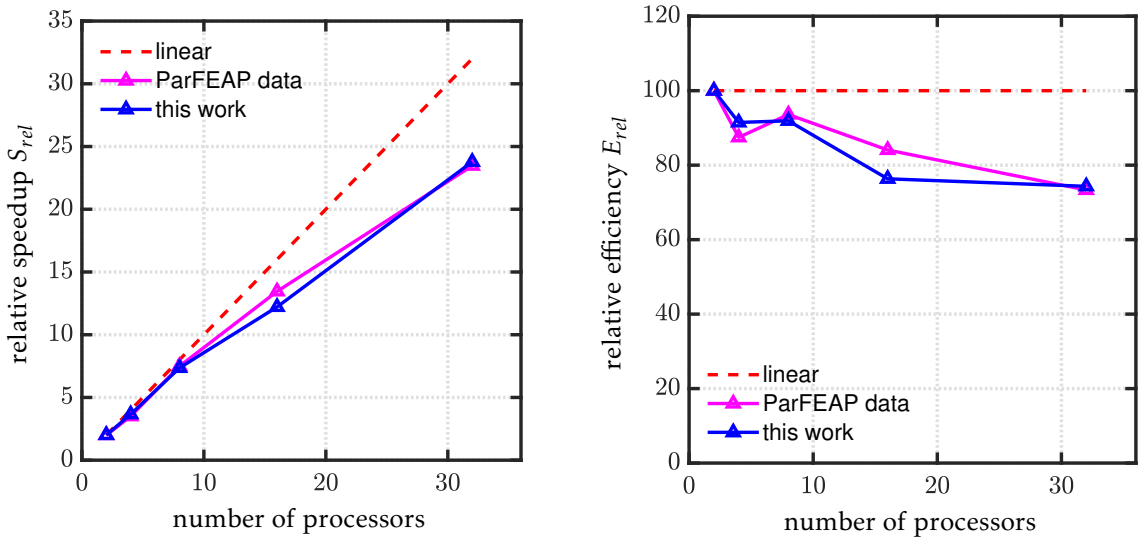


Figure 16: Scalability benchmark of the workstation. The first example of the ParFEAP manual is the analysis of a linear elastic isotropic unit cube. When 32 processors are employed the speedup of the simulation of this work is identical to the speedup given in the manual [28].

of FEAP. As material properties standard values for steel were chosen, E=210 GPa and $\nu = 0.3$.

The performance result is plotted in Fig. 16. Although there are slight differences between the the ParFEAP data and this work's results the speedup values at the upper limit of the range of number of processors are almost identical. The speedup of the simulation of this work is 23.78 compared to the speedup given in the manual of 23.47.

4.1.2 Nonlinear elastic block with 8-node brick elements

The second example that was chosen to validate the scalability of the given ParFEAP program models a nonlinear material response. Again a unit cube was discretized with 125000 8-node brick elements. The neo-Hookean finite deformation model is used with E =210 GPa and ν=0.3. The same boundary conditions are applied to the unit cube as to the examples above. Here the the performance of the simulation of this work reaches 87 % of the speedup of the manual at 32 processors.

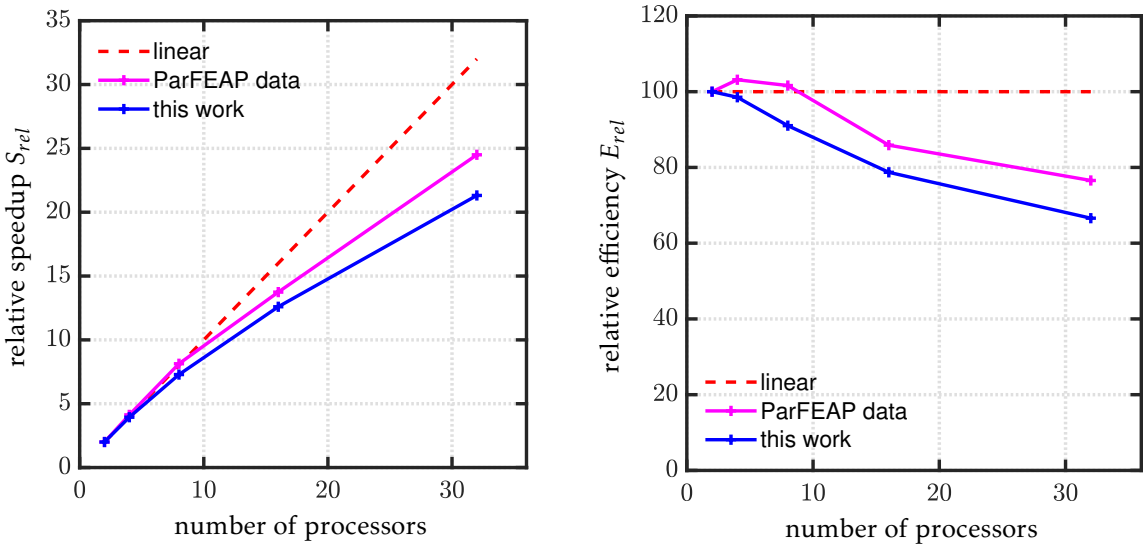


Figure 17: A unit cube of neo-Hookean type of material as second benchmark of the scalability of ParFEAP performed on the given cluster. The speedup reaches 87 % of its reference [28] at 32 processors.

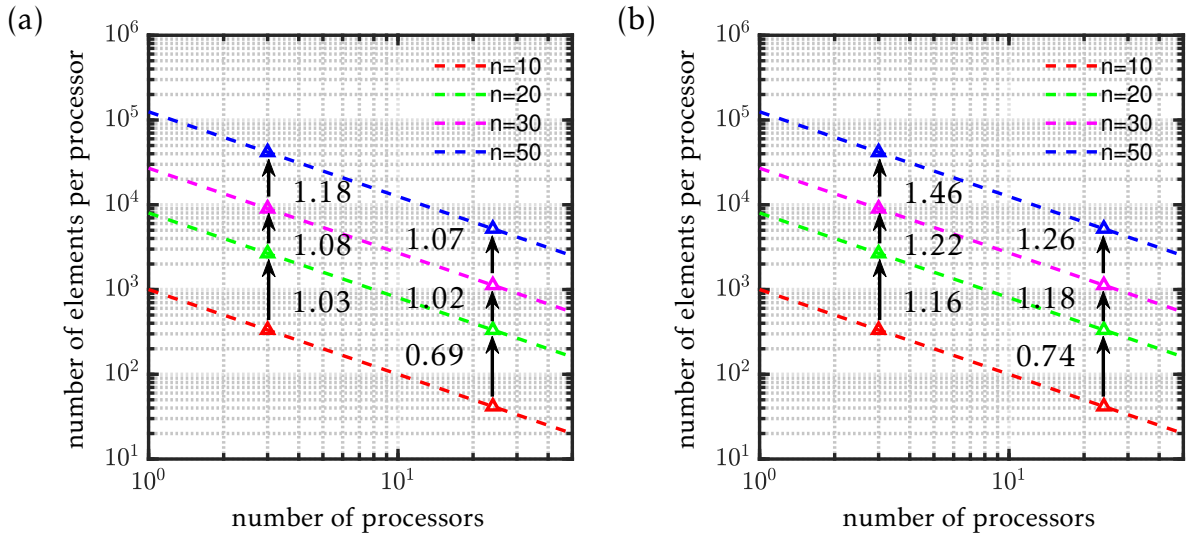


Figure 18: Influence of the problem size on the performance. (a) Linear elastic problem. (b) Fracture.

5 Scalability of PLEANv1.0

5.1 Speedup dependency on the problem size

To understand the speedup behavior of the PLEANv1.0 in more depth following analysis was made. Two problems were modeled and solved on 3 and 24 processors.

A unit cube was discretized with n^3 8-node brick elements. The analysis was made with $n = 10, 20, 30$ and 50 number of elements per edge. A displacement constraint was applied at the bottom at the unit cube. The third degree of freedom of all bottom nodes was clamped. At one corner node the second degree of freedom was fixed additionally and at another corner node the first and second degree of freedom was fixed. At the top of the unit cube a displacement $u = 0.01$ in positive z direction was applied. A fracture problem was considered (see Tab. 7 for material properties) as well as a linear elastic problem for which the yield strength was set to $\sigma_y = 10^{10}$.

The total displacement was applied in 200 load steps. Preconditioner (pc) option pc-type *jacobi* was used (see emphasized note below for further explanation).

The result of the study is shown in Fig. 18. As the number of elements per edge was increased from 10 up to 50 the number of elements per partition was increases by τ. As this requires more computational effort the run time increases by λ. The ratio between both factors is introduced as $f := \frac{\lambda}{\tau}$. This value is given in Fig. 18 for both problems and both series of simulations that were performed on 3 and 24 processors respectively. For the linear elastic problem shown in Fig. 18a the run time increases at a higher rate than the problem size beyond $3 \cdot 10^2$ elements per partition. Yet up to $1 \cdot 10^4$ the run time increases less than 10 % faster. It can be assumed that the reason why f increases with larger partitions is rooted in the cache organization. The fracture problem requires more computational resources and hence f increases at a higher rate than in the case of a linear elastic problem.

This supports the idea of John L. Gustafson who build his speedup law on the assumption (in contrast to Amdahl's Law) that the problem size is not fixed but adapted to the size of the system, i.e. number of processors (see 3.3.2).

5.2 Reference for the measurement of the scalability scope

The PLEANv1.0 is implemented in the parallel FEAP program. A test fracture problem would be analyzed by both the serial and parallel implementation. The comparison of the two speedup plots would give an exact determination of the scalability scope (specifically for the given test case). Following discussion leads to an idea for the scalability scope of the PLEANv1.0. It is based on the comparison of four types of simulation (Tab. 6).

The reference is the scalability of ParFEAP[5] as it is given in [28] and has been used for validation in section 4.1.1. There a linear elastic problem was solved (simulation A). Subsequently the scalability of ParFEAP with the pc *jacobi* becomes the performance reference (simulation B). The serial implementation of the PLEANv1.0 can be compared with ParFEAP with a linear elastic problem, without invoking the fracture problem. Hence the same problem is solved by a serial algorithm on the one hand and on the other hand by a parallel routine (simulation C). Yet there is a slight difference because the phase field means an additional nodal degree of freedom. But because there is no crack initiation the additional serial computation of the phase field can be assumed to be of a negligible amount. If this assumption could not be done the number of nodes or elements would need to be reduced proportionally. Because of the serial computation of the element arrays it is expected that the simulation C will scale less than the simulation of type B.

When a nonlinear elasto-plastic material response as well as fracture is computed simulation D) the above assumption is strongly violated. A significant fraction of computation time would be spent on the serial calculation of the additional internal variables and degrees of freedom. Thus the scalability of simulation D has to be less than the simulation C.

The next step would be the parallelization of the serial PLEANv1.0. An upper bound and an approximation of the lower bound can be made based on the above sequence of argumentation. Clearly the scalability of the parallel PLEANv1.0 has to exceed the scalability of the serial computation (simulation D). Assuming the PLEANv1.0 is entirely parallelized it would reach at least the scalability of ParFEAP when a linear elastic problem is solved (simulation B) depending on the degree of parallelization of ParFEAP's material routine. The parallel computation of element arrays require less communication than the assembly and especially the parallel solution. The percentage of run time regarding the computation of element arrays is higher for the PLEANv1.0 than for ParFEAP since an additional degree of freedom is given. Therefore according to this scenario the parallel PLEANv1.0 implementation could be even more scalable than ParFEAP when a linear elastic isotropic problem is solved.

Thus by means of the simulation with the PLEANv1.0 when neither plasticity nor fracture is invoked the scalability of ParFEAP (when a linear elastic isotropic problem

[5] An important note has to be made on the geometric-algebraic multigrid pc GAMG used to validate the given parallel ParFEAP. Since it is a more efficient preconditioner than jacobi and also designed for multigrid applications it is supposed to yield better performance results than jacobi. Yet the pc jacobi is the only option for problems solved with the PLEANv1.0. For the sake of comparison all the following problems were solved with this pc as well. The PLEANv1.0 program defines 4 degrees of freedom per node. To enable PETSc to handle this specific case the PCSetCoordinates() function inside the usolve.f file would need to be replaced by the MatSetNearNullSpace() function to set the corresponding null space vectors. However, the analyses results presented below show a significantly better scalability of the pc jacobi when compared with GAMG

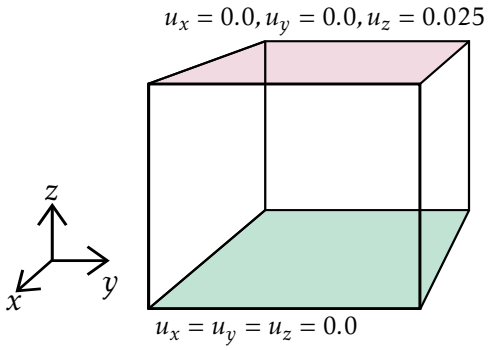


Figure 19: Geometry and boundary conditions of the tensile test.

is solved) can be used as reference of the scalability scope.

simulation	A	B	C	D
program	ParFEAP	ParFEAP	PLEANv1.0	PLEANv1.0
preconditioner	GAMG	jacobi	jacobi	jacobi
material model	linear elastic	linear elastic	linear elastic	fracture
$T_{nproc=3}$ [hh:mm]	04:04	02:31	07:05	16:58

Table 6: Summary of the simulations by which the scalability scope of the PLEANv1.0 is determined.

5.3 Tensile test as problem for scalability study

In all analyses the same tensile test is simulated. A unit cube is discretized with 50^3 elements. The constraints are applied to the unit cube as shown in Fig. 19. Displacement at all bottom nodes are constrained in all spatial directions. Top surface nodes are fixed in x and y direction. In the positive z direction a displacement is prescribed such that a homogeneous strain of 2.5 % is achieved. The prescribed displacement of $u_z = 0.025$ was applied in 200 load steps. The problem was solved with the *conjugated gradient* method. For preconditioning the pc-type was set to *jacobi* except for the scenario A where the pc *GAMG* was used.

For the material models typical values for metals were chosen which are E=206.89 and ν=0.3. The required properties for the linear elastic isotropic material model from FEAP's library were specified as shown in Tab. **??**. This was applied in the cases A and B. To compute a linear elastic material response with the PLEANv1.0 the same values are used as to compute fracture (Tab. 7). Only the yield stress was set to 10^{10} to prohibit the development of plasticity and fracture.

No.	Parameter	name	value	unit
1.	E	Young's modulus	206.89	GPa
2.	μ	shear modulus	80.19	GPa
3.	σ_y	yield stress	300	MPa
4.	σ_∞	saturation stress	450	MPa
5.	κ	bulk modulus	164.2	GPa
6.	α_c	critical equivalent plastic strain	0.005	-
7.	l_d	length scale	0.021	mm
8.	η	viscosity	10^{-7}	GPa · s
9.	h	isotropic hardening modulus	130	MPa
10.	ω	saturation parameter	17	-

Table 7: Material properties used with PLEANv1.0.

In Fig. 20 the result of simulation D is plotted. It shows the development of a typical cross shear band type failure which is observed in J_2-plasticity. As the same result was obtained from all fracture analyses it verifies the correctness of the simulations.

5.4 Parallel performance of PLEANv1.0

All four simulations A, B, C and D were performed on a range of numbers of processors from 3 to 576. Run times were extracted from the log file of each simulation after searching for the maximum value.

At first the absolute speedup plot shall be discussed (Fig. 21). This metric distinguishes itself form the relative speedup as the shortest run time among all programs is chosen as reference (see Section 3.2.3) at the smallest number of processors. Since ParFEAP with jacobi as preconditioner solved the linear elastic problem in the shortest time its run time became the reference for all speedup curves when 3 processors were used. Therefore, its absolute speedup curve is also the best. The PLEANv1.0 required more time to solve the linear elastic problem. Its speedup curve is significantly lower than the one of ParFEAP. The biggest run time was required to solve the fracture problem with the PLEANv1.0. Hence its speedup curve is the lowest one.

That the ParFEAP (pc=jacobi) curve has a negative tangent at the upper range of number of processors and the PLEANv1.0 curves still rise is discussed in more depth in terms of the relative speedup. However the speedup of the PLEANv1.0 is not sufficient to reach the run times of ParFEAP (pc=jacobi).

The more sophisticated preconditioner GAMG scales significantly less than jacobi when used by ParFEAP to solve the linear elastic problem. It stagnates at 48 processors but experiences a slight speedup between 288 and 576 processors.

The absolute run times of the scalability studies are shown in the log-log plot in Fig. 22. Clearly all graphs are proportional to $1/nproc$ up to a certain number of processors but with a different time intercept, i.e. run time when 3 processors are used (exact values are given in Tab. 6). The curve of PLEANv1.0 with fracture starts in a slight nonlinear manner which shows the cache effect which will be discussed. It shows also how the curve of ParFEAP (pc=jacobi) stagnates and rises slightly.

When the speedup up is computed related to the run time at the smallest number of processors of each simulation type the *relative speedup* is calculated. The resulting speedup curves are plotted in Fig. 23. For the fracture problem the partition size when

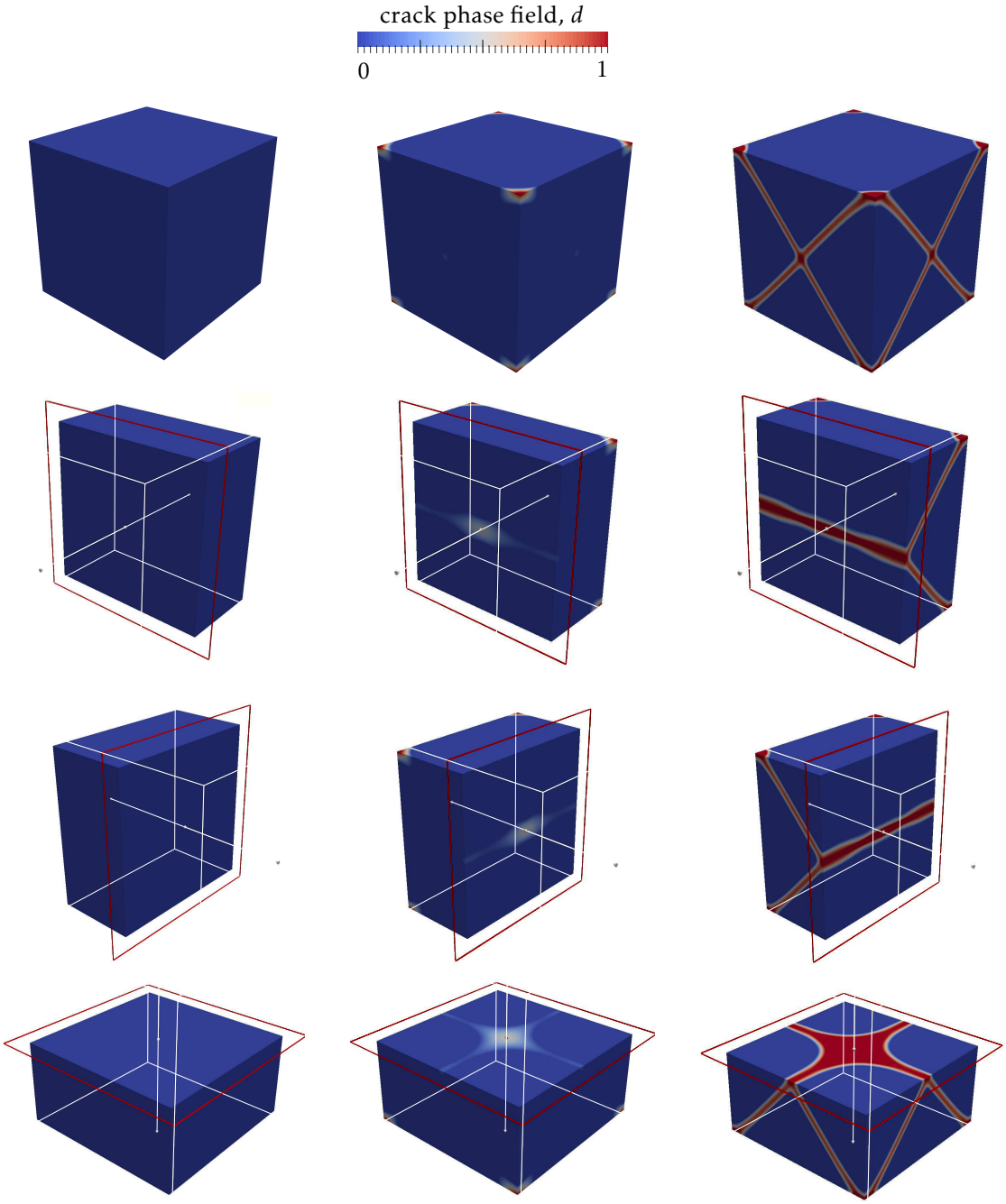


Figure 20: Development of a cross shear band type failure. Left column indicates results at $t = 0$, middle column at $t = T/2$ and right column at $t = T$. The crack initiates at the center and the corners of the cube and evolves to have a cross shape in the x-z and y-z plane. On the vertical surfaces cross shear bands are seen as they typically occur in J_2-plasticity.

only 3 processors are employed exceeds the available cache memory. Would it fit into the cache it could be loaded from there very fast when it is reused. Since that is not the case the data has to be retrieved from main memory always. This increases the run time such that the speedup between 3 and 6 processors is more than theoretically possible. A superlinear speedup is the result which falsifies the speedup curve. Therefore, the lower limit of the range of number of processors is set to 6 instead of 3.

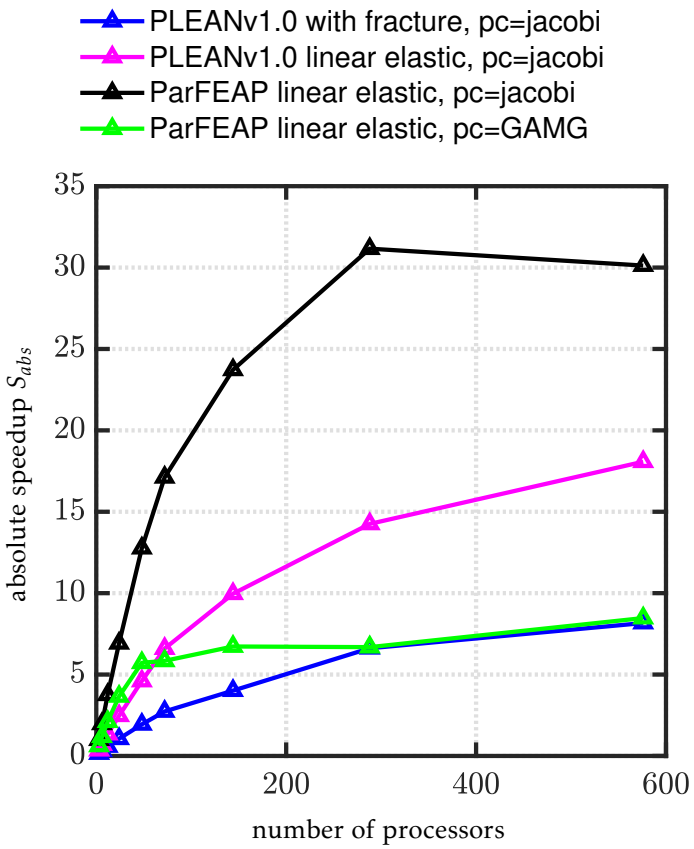


Figure 21: Absolute speedup S_{abs}. The run times of all simulations are related to the fastest program - ParFEAP when a linear elastic program is solved with jacobi as preconditioner - to calculate speedup ratios.

Here the PLEANv1.0 has the best scalability. What can be seen in this plot as well as in the time plot (Fig. 22) is that all curves start with the same slope. As in Section 3.2.2 explained the slope at low numbers of processors shows how big the parallel fraction of an algorithm is since additional costs become effective at higher number of processors. Considering the speedup of all simulations from 6 to 12 processors the speedup of the PLEANv1.0 program solving a fracture problem is 3.8 % less than thePLEANv1.0 and ParFEAP when a linear elastic problem is solved. The speedup of the latter two are identical[6].

The fact that the partition size of the fracture problem when 3 processors are used does not fit into cache is an effect of a significant increase of computational effort. When the PLEANv1.0 program solves a linear elastic problem on 6 processors it requires 2.8 times more run time than ParFEAP. Solving the fracture problem increases the run time by the factor 6.2. Since the PLEANv1.0 program has approximately the same degree of parallelization as ParFEAP the scalability is the same at lower number of processors. At higher numbers of processors it maintains its scalability due to a significantly higher amount of computational effort. In the case of ParFEAP the workload per processor becomes so small that the additional costs exceed the gain through a smaller workload. Therefore the speedup at 576 processors is less than at 288 proces-

[6]Therefore, if the PLEANv1.0 would be parallelized such that it has the same fraction of parallel regions of code as ParFEAP its run time would reduce by 3.5 % at the evaluation point of 12 processors.

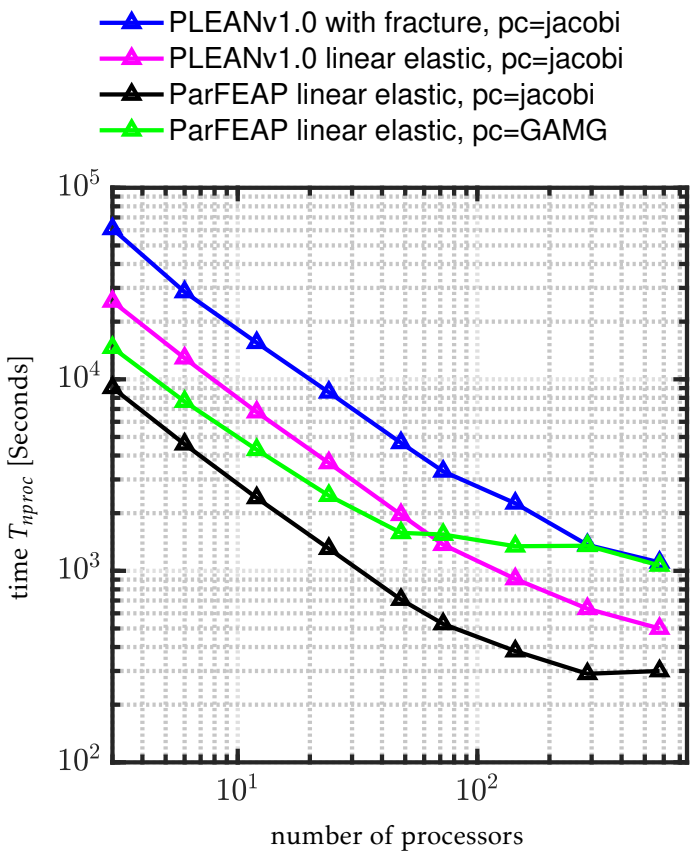


Figure 22: Run times of all analyses in a log-log-plot.

sors. Therefore it can be stated that ParFEAP scales up to 288 processors and reaches its maximum speedup between 288 and 576 processors - probably in the range of 400 processors.

The PLEANv1.0 program on the other hand will reach its maximum speedup beyond 600 processors when it solves a linear elastic problem. When a fracture problem is considered it can be assumed that the maximum speedup is found at a even higher number of processors than when a linear elastic problem is solved. The difference between these two maximums might be significant since the run times of the PLEANv1.0 program solving a fracture problem are 2.2 times greater than in the linear elastic case.

It may be concluded that the scope of the parallelization of the PLEANv1.0 program is negligible. Due to a significantly greater computational effort that the PLEANv1.0 program requires to solve a problem run times are proportionally greater but at the same time it maintains its scalability over a larger range of numbers of processors. It allows an efficient usage of a large parallel system.

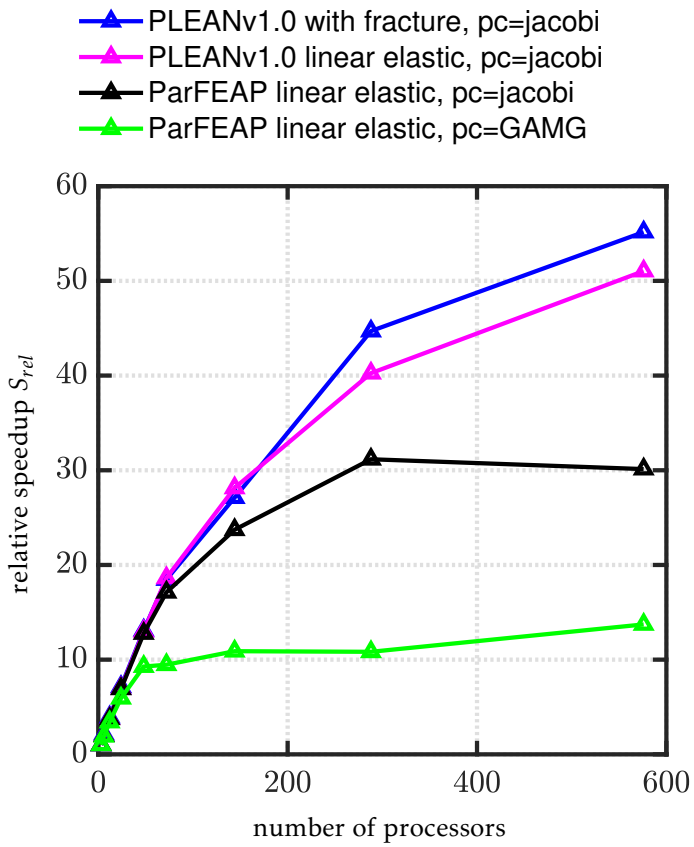


Figure 23: Relative speedup S_{rel}. It shows the scalability of the individual program when a specific problem is solved.

6 Analysis of a Statistically Representative Unit Cell

Based on the promising scalability result for PLEANv1.0 provided by this work, a fracture problem of microstructural unit cell with ca. 10M degrees of freedom will be solved on 600 processors. To approach a physics based simulation on the micro-level a distribution of representative material parameters is considered.

Electron backscatter diffraction (EBSD) data of TiAl was available as a 2 dimensional image that visualizes the structure of roughly 1800 grains and its phases. This image was processed using DREAM.3D[7] to generate a *three dimensional* microstructure with 512 grains as shown in Fig. 24. Considering multiple aspects, e.g. grain size and shape, it can be shown that the generated microstructure is a *statistically representative unit cell* (SRUC). Using DREAM.3D the model is meshed with 11,239,424 elements and 11,390,625 nodes. From the mesh data files are generated that serves as input for PLEANv1.0.

Furthermore, a variation of the yield strength σ_y and the critical equivalent strain α_c over all grains is considered. 100 different values representing a normal distribution with a standard deviation of 5 % are assigned to 512 grains. Fig. 25a and 25b show the histogram plot of the distributions. The same boundary conditions and solution commands as for the scalability study are used (see Sect. 5.3). For analyses of polycrystal microstructure, the most accurate results are obtained under periodic boundary con-

[7] http://dream3d.bluequartz.net/

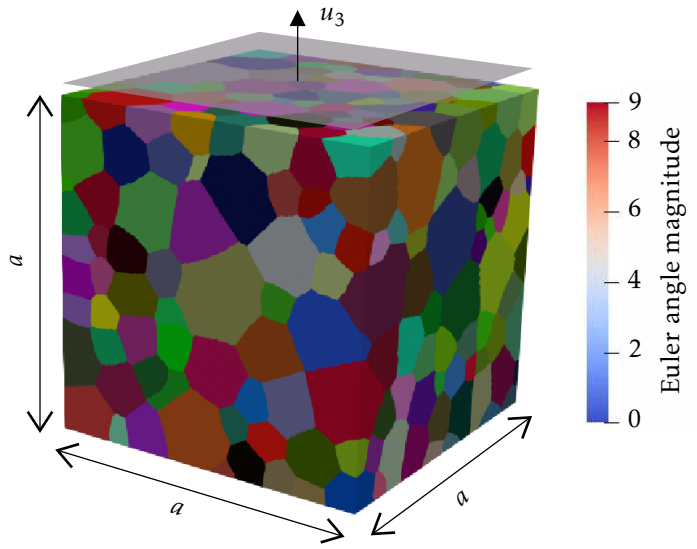


Figure 24: 3D SRUC with 512 grains generated with DREAM.3D on the basis of an EBSD scan.

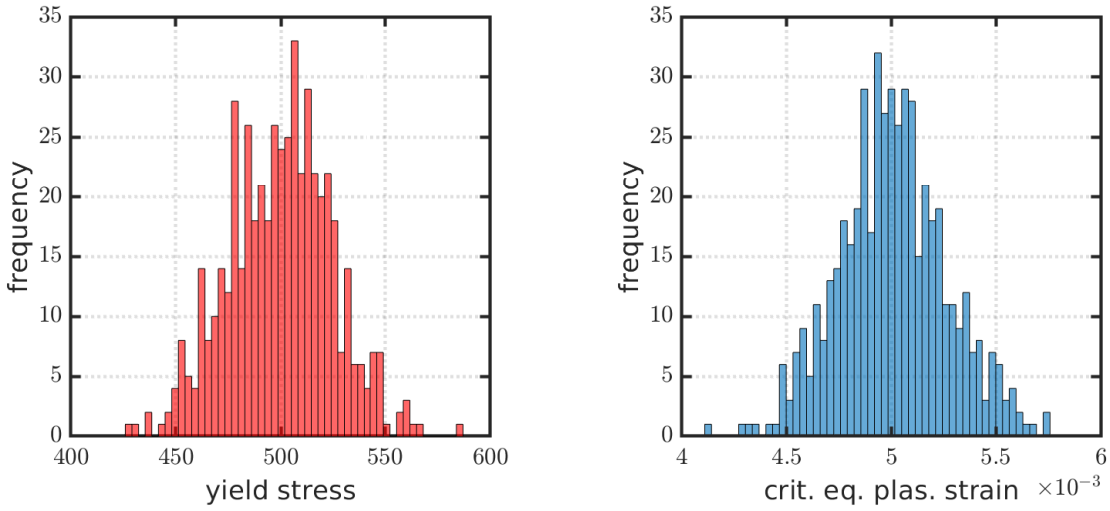


Figure 25: Histogram plots of yield stress and critical equivalent plastic strain.

ditions (PBC) compared to prescribed displacements and forces. On the basis of the strain and stress distribution and the crack initiation and propagation data that will be obtained from this and following analyses further research on physics based large-scale simulations of fracture will be done.

References

[1] Scalability of ANSYS16 applications and Hardware selection. On multi-core and floating point accelerator processor systems. Technical report, ANSYS, Inc., 2018.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings*, 30, 1967. doi: 10.1145/1465482.1465560.

[3] ASM International. *Atlas of Stress-Strain Curves*. ASM International, 2nd edition, December 2002.

[4] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. D. V. Eijkhout, W. Gropp, D. Karpeyev, D. Kaushik, M. Knepley, D. May, L. C. McInnes, R. Mills, T. Munson, K. Rupp, P. Sanan, B. S. S. Zampini, H. Zhang, and H. Zhang. *PETSc Users Manual*, 2018.

[5] J. Dongarra, D. Walker, E. Lusk, B. Knighten, M. Snir, W. Gropp, A. Geist, S. Otto, R. Hempel, J. Cownie, T. Skjellum, L. Clarke, R. Littlefield, M. Sears, and S. Huss-Lederman. *MPI: A Message-Passing Interface Standard. Version 3.1*. MPI Forum, June 2015.

[6] I. Foster. Designing and building parallel programs (online). Website, 2018. URL https://www.mcs.anl.gov/~itf/dbpp/. Addison-Wesley Inc., Argonne National Laboratory, NSF Center for Research on Parallel Computation.

[7] J. L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5): 532–533, May 1986. doi: doi:10.1145/42411.42415.

[8] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 6th edition, November 2017. ISBN (paperback): 9780128119051.

[9] G. Holzapfel. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*. Wiley, 2000. ISBN 9780471823193.

[10] C. Linder and A. Raina. A strong discontinuity approach on multiple levels to model solids at failure. *Computer Methods in Applied Mechanics and Engineering*, 253:558–583, jan 2013. doi: 10.1016/j.cma.2012.07.005.

[11] C. Miehe, F. Welschinger, and M. Hofacker. Thermodynamically consistent phase-filed models of fracture: Variational principles and multi-field FE implementations. *International Journal for Numerical Methods in Engineering*, 83:1273–1311, March 2010. doi: 10.1002/nme.2861.

[12] C. Miehe, F. Aldakheel, and A. Raina. Phase field modeling of ductile fracture at finite strains: a variational gradient-extended plasticity-damage theory. *International Journal of Plasticity*, 84:1–32, sep 2016. doi: 10.1016/j.ijplas.2016.04.011.

[13] D. Negrut. Elements of processor architecture. the hardware/software interplay, 2013.

[14] P. Pacheco. *An Introduction to Parallel Programming*. Elsevier, 1st edition, January 2011.

[15] R. Rabenseifner. Introduction to the message passing interface (mpi), October 2014.

[16] A. Raina. Multi-level descriptions of failure phenom-

ena with the strong discontinuity approach, 2014. URL https://elib.uni-stuttgart.de/handle/11682/542?locale=en.

[17] A. Raina and C. Linder. Modeling crack micro-branching using finite elements with embedded strong discontinuities. *Proceeding in Applied Mathematics and Mechanics*, 10(1):681–684, nov 2010. doi: 10.1002/pamm.201010328.

[18] A. Raina and C. Linder. A strong discontinuity based adaptive refinement approach for the modeling of crack branching. *Proceeding in Applied Mathematics and Mechanics*, 11(1):171–172, dec 2011. doi: 10.1002/pamm.201110077.

[19] A. Raina and C. Linder. Modeling quasi-static crack growth with the embedded finite element method on multiple levels. *Proceeding in Applied Mathematics and Mechanics*, 12(1):135–136, dec 2012. doi: 10.1002/pamm.201210058.

[20] A. Raina and C. Linder. Modeling reorientation phenomena in nonwoven materials with random fiber network microstructure. *Proceeding in Applied Mathematics and Mechanics*, 13(1):249–250, nov 2013. doi: 10.1002/pamm.201310120.

[21] A. Raina and C. Linder. A homogenization approach for nonwoven materials based on fiber undulations and reorientation. *Journal of the Mechanics and Physics of Solids*, 65:12–34, apr 2014. doi: 10.1016/j.jmps.2013.12.011.

[22] A. Raina and C. Linder. Failure in anisotropic nonwoven materials at finite deformation. *Proceeding in Applied Mathematics and Mechanics*, 14(1):377–378, dec 2014. doi: 10.1002/pamm.201410176.

[23] A. Raina and C. Linder. A micromechanical model with strong discontinuities for failure in nonwovens at finite deformations. *International Journal of Solids and Structures*, 75-76:247–259, dec 2015. doi: 10.1016/j.ijsolstr.2015.08.018.

[24] C. N. Richardson, N. Sime, and G. N. Wells. Scalable computation of thermomechanical turbomachinery problems. *Finite Elements in Analysis and Design*, November 2018. doi: doi.org/10.1016/j.finel.2018.11.002.

[25] S. Ristov, R. Prodan, M. Gusev, and K. Skala. Superlinear speedup in hpc systems: why and when? *Annals of Computer Science and Information System*, 2016.

[26] H. R. Schwarz and N. Koeckler. *Numerische Mathematik*. Vieweg+Teubner Verlag, 8th edition, 2011.

[27] R. L. Taylor and S. Govindjee. *FEAP - - A Finite Element Analysis Program. User Manual*. University of California at Berkeley. Department of Civil and Environmental Engineering, 8.5th edition, October 2017.

[28] R. L. Taylor and S. Govindjee. *FEAP - - A Finite Element Analysis Program. Parallel User Manual*. University of California at Berkeley. Department of Civil and Environmental Engineering, 8.5th edition, May 2017.

[29] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier Butterworth-Heinemann, 6th edition, April 2010.